Biometric Systems Private by Design: Reasoning about privacy properties of biometric system architectures

Julien Bringer*, Hervé Chabanne*,**, Daniel Le Métayer***, Roch Lescuyer*

*Idemia, France.
**Télécom ParisTech, Paris, France.
***Inria, Université de Lyon, France.
E-mail: {julien.bringer,herve.chabanne,roch.lescuyer}@idemia.com,
daniel.le-metayer@inria.fr

Abstract. The goal of the work presented in this paper is to show the applicability of the privacy by design approach to biometric systems and the benefit of using formal methods to this end. We build on a general framework for the definition and verification of privacy architectures introduced at STM 2014 and show how it can be adapted to biometrics. The choice of particular techniques and the role of the components (central server, secure module, biometric terminal, smart card, *etc.*) in the architecture have a strong impact on the privacy guarantees provided by a biometric system. Some architectures have already been analysed but on a case by case basis, which makes it difficult to draw comparisons and to provide a rationale for the choice of specific options. In this paper, we describe the application of a general privacy architecture framework to specify different design options for biometric systems and to reason about them in a formal way.

Keywords: Privacy by design, data protection, regulation, formal methods, verification, biometric systems.

1 Introduction

Many applications of biometric recognition have been developed during the last decades in a variety of contexts, from criminal investigations and identity documents to a wealth of public and private usages, like physical access control or smartphone authentication. Biometric systems involve two main phases: enrolment and verification (either authentication or identification) [22]. Enrolment is the registration phase, in which the biometric traits of a person are collected and recorded within the system. During this phase, the identity of the user is

* This work has been partially funded by the French ANR-12-INSE-0013 project BIOPRIV and the European FP7-ICT-2013-1.5 project PRIPARE. Earlier and partial versions of this work appeared in FM 2015 [10] and ISC 2015 [11] conferences. This work provides a global and consistent view of these earlier publications.

associated with his recorded biometric trait. In the authentication mode, a fresh biometric trait is collected and compared with the registered biometric reference to check that it corresponds to the claimed identity. In the identification mode, a fresh biometric data is collected and the corresponding identity is searched in a database of enrolled biometric references. During each phase, to enable efficient and accurate comparison, the collected biometric data are converted into discriminating features, leading to what is called a biometric template.

The increased use of biometric systems has generated a lot of interest in privacy issues and the risks related to biometric trait processing. Since the leakage of biometric traits may lead to serious privacy risks, including tracking and identity theft, it is necessary to follow a privacy by design approach for this type of systems.

The security of biometric systems has been an active research area since at least fifteen years and a wide-array of well-documented primitives have been studied, such as encryption, homomorphic encryption, secure multi-party computation, hardware security or biometric template protection. These building blocks have been used in a variety of privacy preserving biometric systems. Some solutions involve dedicated cryptographic primitives such as secure sketches [14] and fuzzy vaults [23,45], others rely on adaptations of existing cryptographic tools [30] or the use of secure hardware solutions [37]. The choice of particular techniques and the role of the components (central server, secure module, terminal, smart card, *etc.*) in the architecture have a strong impact on the privacy guarantees provided by a solution. However, existing proposals were made on a case by case basis, which makes it difficult to compare them, to provide a rationale for the choice of specific options and to capitalize on past experience.

The objective of the work presented in this paper is to show the applicability of the privacy by design approach to biometric systems and the benefit of using formal methods to this end. We build on a general framework introduced in [2] for the formal definition and validation of privacy architectures. The goal is to specify the various design options in a consistent and comparable way, and then to reason about them in a formal way in order to justify their design in terms of trust assumptions and privacy properties. This work has been conducted within the French ANR research project BioPriv [6], an interdisciplinary project involving lawyers and computer scientists. The BioPriv project itself built on the results of the Turbine European project [1] which studied the best practices for privacy by design applied to biometric systems [27].

The privacy by design approach is often praised by lawyers as well as computer scientists as an essential step towards a better privacy protection. It is now enshrined in the General Data Protection Regulation [36] that will be applied in Europe in May 2018. Nevertheless, it is one thing to impose by law the adoption of privacy by design, quite another to define precisely what it is intended to mean technically-wise and to ensure that it is put into practice by developers. The overall philosophy is that privacy should not be treated as an afterthought but rather as a first-class requirement in the design phase of systems: in other

[1] 7th European Framework Program.

words, designers should have privacy in mind from the start when they define the features and architecture of a system. However, the practical application raises a number of challenges: first of all the privacy requirements must be defined precisely; then it must be possible to reason about potential tensions between privacy and other requirements and to explore different combinations of privacy enhancing technologies to build systems meeting all these requirements.

In Section 2, we provide an outline of the framework introduced in [2] for defining privacy architectures and reasoning about their properties. Then we show how this framework can be used to apply a privacy by design approach to the implementation of biometric systems. In Section 3, we introduce the basic terminology used in this paper and the common features of the biometric architectures considered in the paper. In Section 4, we describe several architectures for biometric systems, considering both existing systems and more advanced solutions, and show that they can be defined in this framework. This makes it possible to highlight their commonalities and differences especially with regard to their underlying trust assumptions.

In the second part of this paper, we address a security issue which cannot be expressed in the framework presented in Section 2. The origin of the problem is that side-channel information may leak from the execution of the system. This issue is acute for biometric systems because the result of a matching between two biometric data inherently provides some information, even if the underlying cryptographic components are correctly implemented [12,40,38]. To adress this issue we propose in Section 5 an extension of the formal framework, in which information leaks spanning over several sessions of the system can be expressed. In Section 6, we apply the extended model to analyse biometric information leakage in several variants of biometric system architectures.

Finally, Section 7 sketches related works and Section 8 concludes the paper with suggestions of avenues for further work.

2 General approach

The work presented in [2] can be seen as a first step towards a formal and systematic approach to privacy by design. In practice, this framework makes it possible to express privacy and integrity requirements (typically the fact that an entity must obtain guarantees about the correctness of a value), to analyse their potential tensions and to make reasoned architectural choices based on explicit trust assumptions. The motivations for the approach come from the following observations:

- First, one of the key decisions that has to be made in the design of a privacy compliant system is the location of the data and the computations: for example, a system in which all data is collected and all results are computed on a central server brings strong integrity guarantees to the operator at the price of a loss of privacy for data subjects. Decentralized solutions may provide better privacy protections but weaker guarantees for the operator. The

use of privacy enhancing technologies such as homomorphic encryption or secure multi-party computation can in some cases reconcile both objectives.
- The choice among the architectural options should be guided by the trust assumptions that can be placed by the actors on the other actors and on the components of the architecture. This trust itself can be justified in different ways (security protocol, secure or certified hardware, accredited third party, *etc.*).

As far as the formal model is concerned, the framework proposed in [2] relies on a dedicated epistemic logic. Indeed, because privacy is closely connected with the notion of knowledge, epistemic logics [16] form an ideal basis to reason about privacy properties. However, standard epistemic logics based on possible worlds semantics suffer from a weakness (called "logical omniscience" [21]) which makes them unsuitable in the context of privacy by design.

We assume that the functionality of the system is expressed as the computation of a set of equations $\Omega := \{X = T\}$ over a language $Term$ of terms T defined as follows, where c represents constants ($c \in Const$), X variables ($X \in Var$) and F functions ($F \in Fun$):

$$T ::= X \mid c \mid F(T_1, \ldots, T_n)$$

An architecture is defined by a set of components C_i, for $i \in [1, N]$, and a set A of relations. The relations define the capacities of the components and the trust assumptions. We use the following language to define the relations:

$$\begin{array}{l} A ::= \{R\} \\ R ::= Has_i(X) \mid Receive_{i,j}(\{St\}, \{X\}) \mid Compute_G(X = T) \\ \qquad\quad \mid Verify_i(St) \qquad\quad \mid Trust_{i,j} \end{array}$$

$$\begin{array}{rl} St ::= Pro \mid & Att \\ Pro ::= Proof_i(\{P\}) & \\ P ::= Att \mid & Eq \end{array} \qquad \begin{array}{rl} Att & ::= Attest_k(\{Eq\}) \\ Eq & ::= Pred(T_1, \ldots, T_m) \end{array}$$

The notation $\{Z\}$ denotes a set of terms of category Z. $Has_i(X)$ denotes the fact that component C_i possesses (or is the origin of) the value of X, which may correspond to situations in which X is stored on C_i or C_i is a sensor collecting the value of X. In this paper, we use the set of predicates $Pred := \{=, \in\}$. $Compute_G(X = T)$ means that the components in the set G can compute the term T and assign its value to X and $Trust_{i,j}$ represents the fact that component C_i trusts component C_j. $Receive_{i,j}(\{St\}, \{X\})$ means that C_i can receive the values of variables in $\{X\}$ together with the statements in $\{St\}$ from C_j .

We consider two types of statements here:

- Attestations: $Attest_k(\{Eq\})$ is the declaration by the component C_k that the properties in $\{Eq\}$ hold.
- Proofs: $Proof_i(\{P\})$ is a proof of properties P provided by C_i.

$Verify_i(St)$ is the verification by component C_i of statement St. If St is a proof statement, $Verify_i(St)$ is the verification of the correctness of St. In contrast, if St is an attestation statement $Attest_k(\{Eq\})$, then $Verify_i(St)$ is the verification of the authenticity of the sender, that is to say C_k. The actual implementation of the relations defining an architecture is not defined at this level. In practice, the verification of an attestation can be implemented as a digital signature verification.

Graphical data flow representations can be derived from architectures expressed in this language. For the sake of readability, we use both notations in the next sections.

The subset of the privacy logic used in this paper is the following dedicated epistemic logic:

$$\begin{aligned} \varphi &::= Has_i(X) \mid Has_i^{none}(X) \mid K_i(Prop) \mid \varphi_1 \ \wedge \ \varphi_2 \\ Prop &::= Pred(T_1, \ldots, T_n) \mid Prop_1 \ \wedge \ Prop_2 \end{aligned}$$

$Has_i(X)$ and $Has_i^{none}(X)$ denote the facts that component C_i respectively can or cannot get the value of X. K_i denotes the epistemic knowledge following the "deductive algorithmic knowledge" philosophy [16,39] that makes it possible to avoid the logical omniscience problem. In this approach, the knowledge of a component C_i is defined as the set of properties that this component can actually derive using its own information and his deductive system $\triangleright_i$.

Another relation, Dep_i, is used to take into account dependencies between variables. $Dep_i(Y, \mathcal{X})$ means that if C_i can obtain the values of each variable in the set of variables $\mathcal{X}$, then it may be able to derive the value of Y. The absence of such a relation is an assumption that C_i cannot derive the value of Y from the values of the variables in $\mathcal{X}$. It should be noted that this dependency relation is associated with a given component: different components may have different capacities. For example, if component C_i is the only component able to decrypt a variable ev to get the clear text v, then $Dep_i(v, \{ev\})$ holds but $Dep_j(v, \{ev\})$ does not hold for any $j \neq i$.

The semantics $S(A)$ of an architecture A is defined as the set of states of the components C_i of A resulting from compatible execution traces [2]. A compatible execution trace contains only events that are instantiations of relations (e.g. $Receive_{i,j}, Compute_G$, *etc.*) of A (as further discussed in Section 5.1). The semantics $S(\varphi)$ of a property φ is defined as the set of architectures meeting φ. For example, $A \in S(Has_i^{none}(X))$ if for all states $\sigma \in S(A)$, the sub-state σ_i of component C_i is such that $\sigma_i(X) = \bot$, which expresses the fact that the component C_i cannot assign a value to the variable X.

To make it possible to reason about privacy properties, an axiomatics of this logic is presented and is proven sound and complete. $A \vdash \varphi$ denotes that φ can be derived from A thanks to the deductive rules (*i.e.* there exists a derivation tree such that all steps belong to the axiomatics, and such that the leaf is $A \vdash \varphi$). A subset of the axioms useful for this paper is presented in Figure 1.

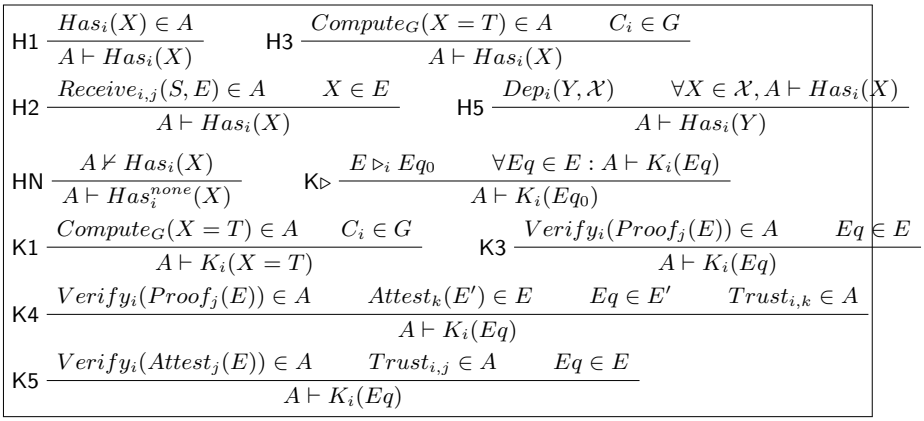


Fig. 1. A subset of rules from the axiomatics of [2]

3 Biometric systems architectures

Before starting the presentation of the different biometric architectures in the next sections, we introduce in this section the basic terminology used in this paper and the common features of the architectures. For the sake of readability, we use upper case sans serif letters S, T, *etc.* as index variables i for components. Type letters `dec`, `br`, *etc.* denote variables. The set of components of an architecture is denoted by $\mathcal{J}$.

The variables used in biometric system architectures are the following:

- A biometric reference template `br` built during the enrolment phase, where a template corresponds to a set or vector of biometrics features that are extracted from raw biometric data in order to be able to compare biometric data accurately.
- A raw biometric data `rd` provided by the user during the verification phase.
- A fresh template `bs` derived from `rd` during the verification phase.
- A threshold `thr` which is used during the verification phase as a closeness criterion for the biometric templates.
- The output `dec` of the verification which is the result of the matching between the fresh template `bs` and the enrolled templates `br`, considering the threshold `thr`.

Two components appear in all biometric architectures: a component U representing the user, and the terminal T which is equipped with a sensor used to acquire biometric traits. In addition, biometric architectures may involve an explicit issuer I, enrolling users and certifying their templates, a server S managing a database containing enrolled templates, a module (which can be a hardware security module, denoted HSM) to perform the matching and eventually to take

the decision, and a smart card C to store the enrolled templates (and in some cases to perform the matching). Figure 2 introduces some graphical representations used in the figures of this paper.

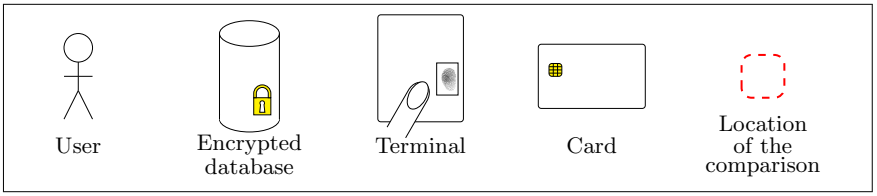


Fig. 2. Graphical representations

In this paper, we focus on the verification phase and assume that enrolment has already been done. Therefore the biometric reference templates are stored on a component which can be either the issuer ($Has_{\mathsf{I}}(\mathtt{br})$) or a smart card ($Has_{\mathsf{C}}(\mathtt{br})$). A verification process is initiated by the terminal T receiving as input a raw biometric data $\mathtt{rd}$ from the user U. T extracts the fresh biometric template $\mathtt{bs}$ from $\mathtt{rd}$ using the function $Extract \in Fun$. All architectures A therefore include $Receive_{\mathsf{T},\mathsf{U}}(\{\}, \{\mathtt{rd}\})$ and $Compute_{\mathsf{T}}(\mathtt{bs} = Extract(\mathtt{rd}))$ and the Dep_{T} relation is such that $(\mathtt{bs}, \{\mathtt{rd}\}) \in Dep_{\mathsf{T}}$. In all architectures A, the user receives the final decision $\mathtt{dec}$ (which can typically be positive or negative) from the terminal: $Receive_{\mathsf{U},\mathsf{T}}(\{\}, \{\mathtt{dec}\}) \in A$. The matching itself, which can be performed by different components depending on the architecture, is expressed by the function $\mu \in Fun$ which takes as arguments two biometric templates and the threshold $\mathtt{thr}$.

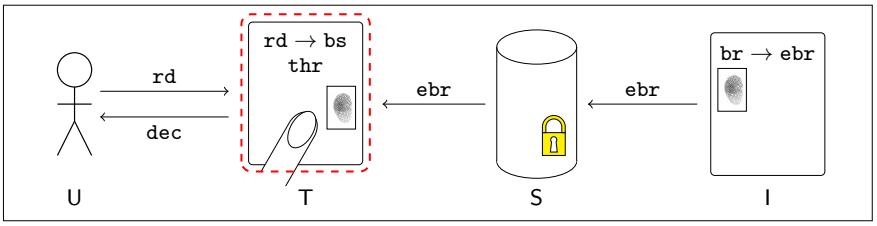
4 Application of the framework to several architectures for biometric systems with various protection levels

In this section, we describe several architectures for biometric systems, considering both existing systems and more advanced solutions, and we show that they can be defined in the framework presented in Section 3.

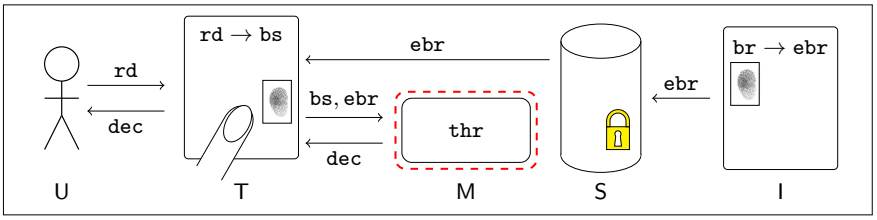
4.1 Protecting the reference templates with encryption

Let us consider first the most common architecture deployed for protecting biometric data. When a user is enrolled his reference template is stored encrypted, either in a terminal with an embedded database, or in a central database. During the identification process, the user supplies a fresh template, the reference templates are decrypted by a component (which can be typically the terminal or a dedicated hardware security module) and the comparison is done inside

this component. The first part of Figure 3 shows an architecture A_{ed} in which reference templates are stored in a central database and the decryption of the references and the matching are done inside the terminal. The second part of the figure shows an architecture A_{hsm} in which the decryption of the references and the matching are done on a dedicated hardware security module. Both architectures are considered in turn in the following paragraphs.



Encrypted database



Encrypted database with a hardware security module (HSM)

Fig. 3. Classical architectures with an encrypted database

While the solutions described in this section can be seen as superseded by the ones of the next section, we introduce them for two purposes: firstly, for a pedagogical reason in order to prepare the reader to better understand the more sophisticated techniques of section 4.2 and secondly, to start with the presentation of in-the-field solutions as they are commonly deployed today.

Use of an encrypted database. The first architecture A_{ed} is composed of a user U, a terminal T, a server S managing an encrypted database $\mathtt{ebr}$ and an issuer I enrolling users and generating the encrypted database $\mathtt{ebr}$. The set Fun includes the encryption and decryption functions Enc and Dec. When applied to an array, Enc is assumed to encrypt each entry of the array. At this stage, for the sake of conciseness, we consider only biometric data in the context of an identification phase. The same types of architectures can be used to deal with authentication, which does not raise any specific issue. The functionality of the architecture is $\Omega := \{\mathtt{ebr} = Enc(\mathtt{br}), \mathtt{br}' = Dec(\mathtt{ebr}), \mathtt{bs} = Extract(\mathtt{rd}),$

$\mathtt{dec} = \mu(\mathtt{br}', \mathtt{bs}, \mathtt{thr})\}$, and the architecture is defined as:

$$\begin{aligned} A_{\mathsf{ed}} := \big\{ & Has_{\mathsf{I}}(\mathtt{br}), Has_{\mathsf{U}}(\mathtt{rd}), Has_{\mathsf{T}}(\mathtt{thr}), Compute_{\mathsf{I}}(\mathtt{ebr} = Enc(\mathtt{br})), \\ & Receive_{\mathsf{S},\mathsf{I}}(\{Attest_{\mathsf{I}}(\mathtt{ebr} = Enc(\mathtt{br}))\}, \{\mathtt{ebr}\}), \\ & Receive_{\mathsf{T},\mathsf{S}}(\{Attest_{\mathsf{I}}(\mathtt{ebr} = Enc(\mathtt{br}))\}, \{\mathtt{ebr}\}), Trust_{\mathsf{T},\mathsf{I}}, \\ & Verify_{\mathsf{T}}(Attest_{\mathsf{I}}(\mathtt{ebr} = Enc(\mathtt{br}))), Receive_{\mathsf{T},\mathsf{U}}(\{\}, \{\mathtt{rd}\}), \\ & Compute_{\mathsf{T}}(\mathtt{bs} = Extract(\mathtt{rd})), Compute_{\mathsf{T}}(\mathtt{br}' = Dec(\mathtt{ebr})), \\ & Compute_{\mathsf{T}}(\mathtt{dec} = \mu(\mathtt{br}', \mathtt{bs}, \mathtt{thr})), Receive_{\mathsf{U},\mathsf{T}}(\{\}, \{\mathtt{dec}\}) \big\} \end{aligned}$$

The properties of the encryption scheme are captured by the dependence and deductive relations. The dependence relations are: $(\mathtt{ebr}, \{\mathtt{br}\}) \in Dep_{\mathsf{I}}$, and $\{(\mathtt{bs}, \{\mathtt{rd}\}), (\mathtt{dec}, \{\mathtt{br}', \mathtt{bs}, \mathtt{thr}\}), (\mathtt{br}', \{\mathtt{ebr}\}), (\mathtt{br}, \{\mathtt{ebr}\})\} \subseteq Dep_{\mathsf{T}}$. Moreover the deductive algorithm relation contains: $\{\mathtt{ebr} = Enc(\mathtt{br})\} \triangleright \{\mathtt{br} = Dec(\mathtt{ebr})\}$.

From the point of view of biometric data protection, the property that this architecture is meant to ensure is the fact that the server should not have access to the reference template, that is to say: $Has^{none}_{\mathsf{S}}(\mathtt{br})$, which can be proven using Rule HN (the same property holds for $\mathtt{br}'$):

$$\mathsf{HN}\ \frac{Has_{\mathsf{S}}(\mathtt{br}) \notin A_{\mathsf{ed}} \quad \nexists \mathcal{X} : (\mathtt{br}, \mathcal{X}) \in Dep_{\mathsf{S}} \quad \nexists T : Compute_{\mathsf{S}}(\mathtt{br} = T) \in A_{\mathsf{ed}} \quad \nexists j \in \mathcal{J}, \nexists S, \nexists E, Receive_{\mathsf{S},j}(S, E) \in A_{\mathsf{ed}} \wedge \mathtt{br} \in E}{A_{\mathsf{ed}} \vdash Has^{none}_{\mathsf{S}}(\mathtt{br})}$$

It is also easy to prove, using $\mathsf{H2}$ and $\mathsf{H5}$, that the terminal has access to $\mathtt{br}'$: $Has_{\mathsf{T}}(\mathtt{br}')$.

As far as integrity is concerned, the terminal should be convinced that the matching is correct. The proof relies on the trust placed by the terminal in the issuer (about the correctness of $\mathtt{ebr}$) and the computations that the terminal can perform by itself (through $Compute_{\mathsf{T}}$ and the application of $\triangleright$):

$$\mathsf{K5}\ \frac{Verify_{\mathsf{T}}(\{Attest_{\mathsf{I}}(\mathtt{ebr} = Enc(\mathtt{br}))\}) \in A_{\mathsf{ed}} \qquad Trust_{\mathsf{T},\mathsf{I}} \in A_{\mathsf{ed}}}{A_{\mathsf{ed}} \vdash K_{\mathsf{T}}(\mathtt{ebr} = Enc(\mathtt{br}))}$$

$$\mathsf{K}\triangleright\ \frac{\{\mathtt{ebr} = Enc(\mathtt{br})\} \triangleright \{\mathtt{br} = Dec(\mathtt{ebr})\} \qquad A_{\mathsf{ed}} \vdash K_{\mathsf{T}}(\mathtt{ebr} = Enc(\mathtt{br}))}{A_{\mathsf{ed}} \vdash K_{\mathsf{T}}(\mathtt{br} = Dec(\mathtt{ebr}))}$$

$$\mathsf{K1}\ \frac{Compute_{\mathsf{T}}(\mathtt{br}' = Dec(\mathtt{ebr})) \in A_{\mathsf{ed}}}{A_{\mathsf{ed}} \vdash K_{\mathsf{T}}(\mathtt{br}' = Dec(\mathtt{ebr}))}$$

Assuming that all deductive relations include the properties (commutativity and transitivity) of the equality, $\mathsf{K}\triangleright$ can be used to derive: $A_{\mathsf{ed}} \vdash K_{\mathsf{T}}(\mathtt{br} = \mathtt{br}')$. A further application of $\mathsf{K1}$ with another transitivity rule for the equality allows us to obtain the desired integrity property:

$$\mathsf{K}\triangleright\ \frac{A_{\mathsf{ed}} \vdash K_{\mathsf{T}}(\mathtt{br} = \mathtt{br}') \qquad \mathsf{K1}\ \dfrac{Compute_{\mathsf{T}}(\mathtt{dec} = \mu(\mathtt{br}', \mathtt{bs}, \mathtt{thr})) \in A_{\mathsf{ed}}}{A_{\mathsf{ed}} \vdash K_{\mathsf{T}}(\mathtt{dec} = \mu(\mathtt{br}', \mathtt{bs}, \mathtt{thr}))}}{A_{\mathsf{ed}} \vdash K_{\mathsf{T}}(\mathtt{dec} = \mu(\mathtt{br}, \mathtt{bs}, \mathtt{thr}))}$$

Encrypted database with a hardware security module. The architecture presented in the previous subsection relies on the terminal to decrypt the reference template and to perform the matching operation. As a result, the clear reference template is known by the terminal and the only component that has to be trusted by the terminal is the issuer. If it does not seem sensible to entrust the terminal with this central role, another option is to delegate the decryption of the reference template and computation of the matching to a hardware security module so that the terminal itself never stores any clear reference template. This strategy leads to architecture A_{hsm} pictured in the second part of Figure 3.

In addition to the user U, the issuer I, the terminal T, and the server S, the set of components contains a hardware security module M. The terminal does not perform the matching, but has to trust M. This trust can be justified in practice by the level of security provided by the HSM M (which can also be endorsed by an official security certification scheme). The architecture is described as follows in our framework:

$$\begin{aligned} A_{\mathsf{hsm}} := \big\{ & Has_{\mathsf{I}}(\mathtt{br}), Has_{\mathsf{U}}(\mathtt{rd}), Has_{\mathsf{M}}(\mathtt{thr}), Compute_{\mathsf{I}}(\mathtt{ebr} = Enc(\mathtt{br})), \\ & Receive_{\mathsf{S},\mathsf{I}}(\{Attest_{\mathsf{I}}(\mathtt{ebr} = Enc(\mathtt{br}))\}, \{\mathtt{ebr}\}), \\ & Receive_{\mathsf{T},\mathsf{S}}(\{Attest_{\mathsf{I}}(\mathtt{ebr} = Enc(\mathtt{br}))\}, \{\mathtt{ebr}\}), Trust_{\mathsf{T},\mathsf{I}}, \\ & Verify_{\mathsf{T}}(Attest_{\mathsf{I}}(\mathtt{ebr} = Enc(\mathtt{br}))), Receive_{\mathsf{T},\mathsf{U}}(\{\}, \{\mathtt{rd}\}), \\ & Compute_{\mathsf{T}}(\mathtt{bs} = Extract(\mathtt{rd})), Receive_{\mathsf{M},\mathsf{T}}(\{\}, \{\mathtt{bs}, \mathtt{ebr}\}), \\ & Compute_{\mathsf{M}}(\mathtt{br}' = Dec(\mathtt{ebr})), Compute_{\mathsf{M}}(\mathtt{dec} = \mu(\mathtt{br}', \mathtt{bs}, \mathtt{thr})), \\ & Verify_{\mathsf{T}}(\{Attest_{\mathsf{M}}(\mathtt{dec} = \mu(\mathtt{br}', \mathtt{bs}, \mathtt{thr}))\}), Trust_{\mathsf{T},\mathsf{M}}, \\ & Receive_{\mathsf{T},\mathsf{M}}(\mathcal{A}, \{\mathtt{dec}\}), Verify_{\mathsf{T}}(\{Attest_{\mathsf{M}}(\mathtt{br}' = Dec(\mathtt{ebr}))\}) \big\} \end{aligned}$$

where the set of attestations $\mathcal{A}$ received by the terminal from the module is $\mathcal{A} := \{Attest_{\mathsf{M}}(\mathtt{dec} = \mu(\mathtt{br}', \mathtt{bs}, \mathtt{thr})), Attest_{\mathsf{M}}(\mathtt{br}' = Dec(\mathtt{ebr}))\}$.

The trust relation between the terminal and the module makes it possible to apply rule $\mathsf{K5}$ twice:

$$\mathsf{K5}\ \frac{Verify_{\mathsf{T}}(\{Attest_{\mathsf{M}}(\mathtt{dec} = \mu(\mathtt{br}', \mathtt{bs}, \mathtt{thr}))\}) \in A_{\mathsf{hsm}} \qquad Trust_{\mathsf{T},\mathsf{M}} \in A_{\mathsf{hsm}}}{A_{\mathsf{hsm}} \vdash K_{\mathsf{T}}(\mathtt{dec} = \mu(\mathtt{br}', \mathtt{bs}, \mathtt{thr}))}$$

$$\mathsf{K5}\ \frac{Verify_{\mathsf{T}}(\{Attest_{\mathsf{M}}(\mathtt{br}' = Dec(\mathtt{ebr}))\}) \in A_{\mathsf{hsm}} \qquad Trust_{\mathsf{T},\mathsf{M}} \in A_{\mathsf{hsm}}}{A_{\mathsf{hsm}} \vdash K_{\mathsf{T}}(\mathtt{br}' = Dec(\mathtt{ebr}))}$$

The same proof as in the previous subsection can be applied to establish the integrity of the matching. The trust relation between the terminal and the issuer and the rules $\mathsf{K5}$, $\mathsf{K}\triangleright$ make it possible to derive: $A_{\mathsf{hsm}} \vdash K_{\mathsf{T}}(\mathtt{br} = Dec(\mathtt{ebr}))$. Then two successive applications of $\mathsf{K}\triangleright$ regarding the transitivity of the equality lead to: $A_{\mathsf{hsm}} \vdash K_{\mathsf{T}}(\mathtt{dec} = \mu(\mathtt{br}, \mathtt{bs}, \mathtt{thr}))$.

As in architecture A_{ed}, the biometric references are never disclosed to the server. However, in contrast with A_{ed}, they are not disclosed either to the terminal, as shown by rule HN:

$$\mathsf{HN}\ \frac{Has_{\mathsf{T}}(\mathtt{br}) \notin A_{\mathsf{hsm}} \quad \nexists \mathcal{X} : (\mathtt{br}, \mathcal{X}) \in Dep_{\mathsf{T}} \quad \nexists T : Compute_{\mathsf{T}}(\mathtt{br} = T) \in A_{\mathsf{hsm}} \quad \nexists j \in \mathcal{J}, \nexists S, \nexists E, Receive_{\mathsf{T},j}(S, E) \in A_{\mathsf{hsm}} \wedge \mathtt{br} \in E}{A_{\mathsf{hsm}} \vdash Has_{\mathsf{T}}^{none}(\mathtt{br})}$$

4.2 Enhancing protection with homomorphic encryption

In both architectures of Section 4.1, biometric templates are protected, but the component performing the matching (either the terminal or the secure module) gets access to the reference templates. In this section, we show how homomorphic encryption can be used to ensure that no component gets access to the biometric reference templates during the verification.

Homomorphic encryption schemes [18] makes it possible to compute certain functions over encrypted data. For example, if Enc is a homomorphic encryption scheme for multiplication then there is an operation $\otimes$ such that:

$$c_1 = Enc(m_1) \wedge c_2 = Enc(m_2) \Rightarrow c_1 \otimes c_2 = Enc(m_1 \times m_2).$$

Figure 4 presents an architecture A_{hom} derived from A_{hsm} in which the server performs the whole matching computation over encrypted data. The user supplies a template that is sent encrypted to the server (denoted `ebs`). The server also owns an encrypted reference template `ebr`. The comparison, i.e. the computation of the distance between the templates, is done by the server, leading to the encrypted distance `edec`, but the server does not get access to the biometric data or to the result. This is made possible through the use a homomorphic encryption scheme. On the other hand, the module gets the result, but does not get access to the templates. Let us note that A_{hom} is just one of the possible ways to use homomorphic encryption in this context: the homomorphic computation of the distance could actually be made by another component (for example the terminal itself) since it does not lead to any leak of biometric data.

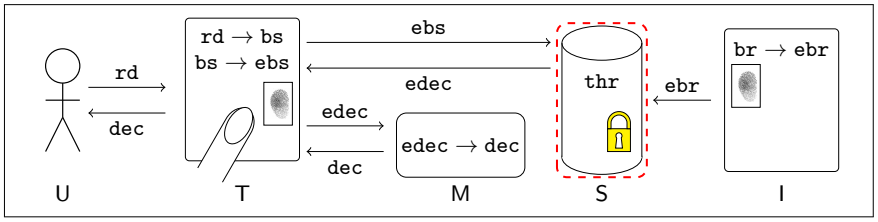


Fig. 4. Comparison over encrypted data with homomorphic encryption

The homomorphic property of the encryption scheme needed for this application depends on the matching algorithm. An option is to resort to a fully homomorphic encryption scheme (FHE) [18] as in the solution described in [44] which uses a variant of a FHE scheme for face-recognition. However, schemes

with simpler homomorphic functionalities can also be sufficient (examples can be found in [8,7]). Since we describe our solutions at the architecture level, we do not need to enter into details regarding the chosen homomorphic scheme. We just need to assume the existence of a homomorphic matching function $Hom\text{-}\mu$ with the following properties captured by the algorithmic knowledge relations:

$$\{\mathtt{ebr} = Enc(\mathtt{br}), \mathtt{ebs} = Enc(\mathtt{bs}), \mathtt{edec} = Hom\text{-}\mu(\mathtt{ebr}, \mathtt{ebs}, \mathtt{thr})\} \triangleright \{Dec(\mathtt{edec}) = \mu(\mathtt{br}, \mathtt{bs}, \mathtt{thr})\} \quad (1)$$

The dependence relations include the following: $\{(\mathtt{bs}, \{\mathtt{rd}\}), (\mathtt{ebs}, \{\mathtt{bs}\})\} \subseteq Dep_{\mathsf{T}}$; $(\mathtt{ebr}, \{\mathtt{br}\}) \in Dep_{\mathsf{I}}$; $\{(\mathtt{br}, \{\mathtt{ebr}\}), (\mathtt{bs}, \{\mathtt{ebs}\}), (\mathtt{dec}, \{\mathtt{edec}\})\} \subseteq Dep_{\mathsf{M}}$. Architecture A_{hom} is defined as follows:

$$\begin{aligned} A_{\mathsf{hom}} := \big\{ & Has_{\mathsf{I}}(\mathtt{br}), Has_{\mathsf{U}}(\mathtt{rd}), Has_{\mathsf{S}}(\mathtt{thr}), Compute_{\mathsf{I}}(\mathtt{ebr} = Enc(\mathtt{br})), \\ & Receive_{\mathsf{S},\mathsf{I}}(\{Attest_{\mathsf{I}}(\{\mathtt{ebr} = Enc(\mathtt{br})\})\}, \{\mathtt{ebr}\}), Receive_{\mathsf{T},\mathsf{U}}(\{\}, \{\mathtt{rd}\}), \\ & Compute_{\mathsf{T}}(\mathtt{bs} = Extract(\mathtt{rd})), Compute_{\mathsf{T}}(\mathtt{ebs} = Enc(\mathtt{bs})), \\ & Receive_{\mathsf{S},\mathsf{T}}(\{\}, \{\mathtt{ebs}\}), Compute_{\mathsf{S}}(\mathtt{edec} = Hom\text{-}\mu(\mathtt{ebr}, \mathtt{ebs}, \mathtt{thr})), \\ & Receive_{\mathsf{T},\mathsf{S}}(\mathcal{A}, \{\mathtt{edec}\}), Verify_{\mathsf{T}}(Attest_{\mathsf{I}}(\{\mathtt{ebr} = Enc(\mathtt{br})\})), \\ & Verify_{\mathsf{T}}(Attest_{\mathsf{S}}(\{\mathtt{edec} = Hom\text{-}\mu(\mathtt{ebr}, \mathtt{ebs}, \mathtt{thr})\})), Trust_{\mathsf{T},\mathsf{S}}, \\ & Trust_{\mathsf{T},\mathsf{I}}, Receive_{\mathsf{M},\mathsf{T}}(\{\}, \{\mathtt{edec}\}), Compute_{\mathsf{M}}(\mathtt{dec} = Dec(\mathtt{edec})), \\ & Receive_{\mathsf{T},\mathsf{M}}(\{Attest_{\mathsf{M}}(\{\mathtt{dec} = Dec(\mathtt{edec})\})\}, \{\mathtt{dec}\}), Trust_{\mathsf{T},\mathsf{M}}, \\ & Verify_{\mathsf{T}}(Attest_{\mathsf{M}}(\{\mathtt{dec} = Dec(\mathtt{edec})\})), Receive_{\mathsf{U},\mathsf{T}}(\{\}, \{\mathtt{dec}\})\big\} \end{aligned}$$

where the set $\mathcal{A}$ of attestations received by the terminal from the server is: $\mathcal{A} := \{Attest_{\mathsf{I}}(\{\mathtt{ebr} = Enc(\mathtt{br})\}), Attest_{\mathsf{S}}(\{\mathtt{edec} = Hom\text{-}\mu(\mathtt{ebr}, \mathtt{ebs}, \mathtt{thr})\})\}$.

In order to prove that the terminal can establish the integrity of the result `dec`, we can proceed in two steps, proving first the correctness of `edec` and then deriving the correctness of `edec` using the properties of homomorphic encryption. The first step relies on the capacities of component T and the trust assumptions on components I and S using rules $\mathsf{K1}$ and $\mathsf{K5}$ respectively.

$$\mathsf{K1}\ \frac{Compute_{\mathsf{T}}(\mathtt{ebs} = Enc(\mathtt{bs})) \in A_{\mathsf{hom}}}{A_{\mathsf{hom}} \vdash K_{\mathsf{T}}(\mathtt{ebs} = Enc(\mathtt{bs}))}$$

$$\mathsf{K5}\ \frac{Verify_{\mathsf{T}}(\{Attest_{\mathsf{I}}(\mathtt{ebr} = Enc(\mathtt{br}))\}) \in A_{\mathsf{hom}} \qquad Trust_{\mathsf{T},\mathsf{I}} \in A_{\mathsf{hom}}}{A_{\mathsf{hom}} \vdash K_{\mathsf{T}}(\mathtt{ebr} = Enc(\mathtt{br}))}$$

$$\mathsf{K5}\ \frac{Verify_{\mathsf{T}}(\{Attest_{\mathsf{S}}(\mathtt{edec} = Hom\text{-}\mu(\mathtt{br}, \mathtt{bs}, \mathtt{thr}))\}),\ Trust_{\mathsf{T},\mathsf{S}} \in A_{\mathsf{hom}}}{A_{\mathsf{hom}} \vdash K_{\mathsf{T}}(\mathtt{edec} = Hom\text{-}\mu(\mathtt{br}, \mathtt{bs}, \mathtt{thr}))}$$

The second step can be done through the application of the deductive algorithmic knowledge regarding the homomorphic encryption property (with LHS_1 the left hand-side of equation (1)) :

$$\mathsf{K}\triangleright\ \frac{LHS_1 \triangleright \{Dec(\mathtt{edec}) = \mu(\mathtt{br}, \mathtt{bs}, \mathtt{thr})\} \qquad \forall Eq \in LHS_1 : A_{\mathsf{hom}} \vdash K_{\mathsf{T}}(Eq)}{A_{\mathsf{hom}} \vdash K_{\mathsf{T}}(Dec(\mathtt{edec}) = \mu(\mathtt{br}, \mathtt{bs}, \mathtt{thr}))}$$

The desired property is obtained through the application of rules $\mathsf{K5}$ and $\mathsf{K}\triangleright$ exploiting the trust relation between T and M and the transitivity of equality.

$$\mathsf{K5}\ \frac{Verify_{\mathsf{T}}(\{Attest_{\mathsf{M}}(\mathtt{dec} = Dec(\mathtt{edec}))\}) \in A_{\mathsf{hom}} \qquad Trust_{\mathsf{T},\mathsf{M}} \in A_{\mathsf{hom}}}{A_{\mathsf{hom}} \vdash K_{\mathsf{T}}(\mathtt{dec} = Dec(\mathtt{edec}))}$$

$$\mathsf{K}\triangleright\ \frac{A_{\mathsf{hom}} \vdash K_{\mathsf{T}}(Dec(\mathtt{edec}) = \mu(\mathtt{br},\mathtt{bs},\mathtt{thr})) \quad A_{\mathsf{hom}} \vdash K_{\mathsf{T}}(\mathtt{dec} = Dec(\mathtt{edec}))}{A_{\mathsf{hom}} \vdash K_{\mathsf{T}}(\mathtt{dec} = \mu(\mathtt{br},\mathtt{bs},\mathtt{thr}))}$$

As far as privacy is concerned, the main property that A_{hom} is meant to ensure is that no component (except the issuer) has access to the biometric references. Rule HN makes it possible to prove that U, T, and S never get access to $\mathtt{br}$, as in Section 4.1. The same rule can be applied here to prove $A_{\mathsf{hom}} \nvdash Has_{\mathsf{M}}(\mathtt{ebr})$ exploiting the fact that neither $(\mathtt{br}, \{\mathtt{edec}\})$ nor $(\mathtt{br}, \{\mathtt{dec}\})$ belong to Dep_{M}.

4.3 The Match-On-Card technology

Another solution can be considered when the purpose of the system is authentication rather than identification. In this case, it is not necessary to store a database of biometric reference templates and a (usually unique) reference template can be stored on a smart card. A smart card based privacy preserving architecture has been proposed recently which relies on the idea of using the card not only to store the reference template but also to perform the matching itself. Since the comparison is done inside the card the reference template never leaves the card. In this *Match-On-Card* (MOC) technology [37,35,19] (also called *comparison-on-card*), the smart card receives the fresh biometric template, carries out the comparison with its reference template, and sends the decision back (as illustrated in Figure 5).

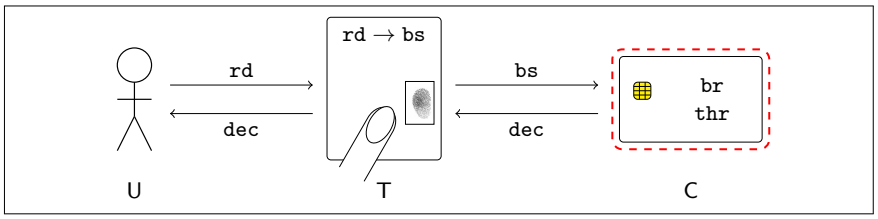


Fig. 5. Biometric verification using the Match-On-Card technology

In this architecture, the terminal is assumed to trust the smart card. This trust assumption is justified by the fact that the card is a tamper-resistant hardware element. This architecture is simpler than the previous ones but not always possible in practice (for a combination of technical and economic reasons) and may represent a shift in terms of trust if the smart card is under the control of the user.

More formally, the MOC architecture is composed of a user U, a terminal T, and a card C. The card C attests that the templates $\mathtt{br}$ and $\mathtt{bs}$ are close (with respect to the threshold $\mathtt{thr}$):

$$\begin{aligned} A_{\mathsf{moc}} := \big\{ & Has_{\mathsf{C}}(\mathtt{br}), Has_{\mathsf{U}}(\mathtt{rd}), Has_{\mathsf{C}}(\mathtt{thr}), Receive_{\mathsf{T},\mathsf{U}}(\{\},\{\mathtt{rd}\}), \\ & Compute_{\mathsf{T}}(\mathtt{bs} = Extract(\mathtt{rd})), Receive_{\mathsf{C},\mathsf{T}}(\{\},\{\mathtt{bs}\}), \\ & Compute_{\mathsf{C}}(\mathtt{dec} = \mu(\mathtt{br},\mathtt{bs},\mathtt{thr})), Receive_{\mathsf{U},\mathsf{T}}(\{\},\{\mathtt{dec}\}), \\ & Receive_{\mathsf{T},\mathsf{C}}(\{Attest_{\mathsf{C}}(\mathtt{dec} = \mu(\mathtt{br},\mathtt{bs},\mathtt{thr}))\},\{\mathtt{dec}\}), \\ & Verify_{\mathsf{T}}(\{Attest_{\mathsf{C}}(\mathtt{dec} = \mu(\mathtt{br},\mathtt{bs},\mathtt{thr}))\}), Trust_{\mathsf{T},\mathsf{C}} \big\} \end{aligned}$$

Using rule HN, it is easy to show that no component apart from C gets access to $\mathtt{br}$. The proof of the integrity property relies on the capacities of component T and the trust assumption on component C using rules $\mathsf{K1}$ and $\mathsf{K5}$ respectively.

5 Extension of the framework to information leakage

In this section, we address a security issue which cannot be expressed in the framework presented in Section 2. The origin of the problem is that side-channel information may leak from the execution of the system. This issue is acute for biometric systems because the result of a matching between two biometric data inherently provides some information, even if the underlying cryptographic components are correctly implemented [12,40,38]. To address this issue we propose in Section 5.1 an extension of the architecture language and in Section 5.2 an extension of the privacy logic.

5.1 Extension of the architecture language

Motivated by the need to analyse the inherent leakage of the result of a matching between two biometric data in biometric systems (cf. [12,40,38]), we now propose an extension of the formal framework sketched in Section 2, in which the information leaking through several executions can be expressed. In fact, in this line of work, it is shown that someone might mount an hill-climbing attack where he is trying, by repeated trials, to guess the templates that are privately stored in a secure component. Doing that, he will exploit the information leaked during one execution, e.g. the similarity result between his guess and the templates. At the end, his strategy is to increase his knowledge of the templates, execution after execution. We thus have to extend our static model used in previous sections, to handle this more dynamic situation.

We highlight the difference with the framework introduced in Section 2 without repeating their common part. The term language we use is now the following.

$$\begin{aligned} T &::= \tilde{X} \mid \quad c \quad \mid F(\tilde{X}_1, \ldots, \tilde{X}_m, c_1, \ldots, c_q) \\ \tilde{X} &::= X \mid X[k] \end{aligned}$$

Functions may take as parameters both variables and constants. Variables $\tilde{X}$ can be simple variables or arrays of variables. If X is an array, $Range(X)$ denotes its size.

In this extended framework, in addition to defining a set of primitives, an architecture can also provide a bound on the number of times a primitive can be used.

$$
\begin{array}{l}
A ::= \{R\} \\
R ::= Has_i^{(n)}(X) \mid Has_i(c) \mid Receive_{i,j}^{(n)}(\{St\}, \{X\} \cup \{c\}) \\
\qquad \mid Trust_{i,j} \quad \mid Reset \quad \mid Compute_G^{(n)}(X = T) \mid Verify_i^{(n)}(\{St\})
\end{array}
$$

$$
\begin{array}{rlrl}
St ::= & Pro \mid Att & Att ::= & Attest_i(\{Eq\}) \\
Pro ::= & Proof_i(\{P\}) & Eq ::= & Pred(T_1, \ldots, T_m) \\
P ::= & Att \mid Eq &&
\end{array}
$$

The superscript notation $^{(n)}$ denotes that a primitive can be carried out at most $n \in (\mathbb{N} \setminus \{0\}) \cup \{\infty\}$ times by the component(s) – where $(\forall n' \in \mathbb{N}\colon n' < \infty)$. We assume that n is never equal to 0. $\mathsf{mul}(\alpha)$ denotes the multiplicity (n) of the primitive α, if any. The *Reset* primitive is used to reinitialize the whole system.

As in the initial model, consistency assumptions are made about the architectures to avoid meaningless definitions. For instance, we require that components carry out computations only on the values that they have access to (either through Has, $Compute$, or $Receive$). We also require that all multiplicities n specified by the primitives are identical in a consistent architecture. As a result, a consistent architecture A is parametrized by an integer $n \geq 1$ (we note $A(n)$ when we want to make this integer explicit).

A key concept for the definition of the semantics is the notion of trace. A trace is a sequence of events and an event[2] is an instantiation of an architectural primitive[3]. The notion of successive sessions is caught by the addition of a $Session$ event[4] . A trace θ of events is said compatible with a consistent architecture $A(n)$ if all events in θ (except the computations) can be obtained by instantiation of some architectural primitive from A, and if the number of events between two *Reset* events corresponding to a given primitive is less than the bound n specified by the architecture. We denote by $T(A)$ the set of traces which are compatible with an architecture A.

$$
\begin{array}{l}
\theta ::= \mathsf{Seq}(\epsilon) \\
\epsilon ::= Has_i(X : V) \mid Has_i(c) \mid Receive_{i,j}(\{St\}, \{X : V\} \cup \{c\}) \\
\qquad \mid Session \quad \mid Reset \quad \mid Compute_G(X = T) \mid Verify_i(\{St\})
\end{array}
$$

An event can instantiate variables X with specific values V. Constants always map to the same value. Let Val be the set of values the variables and constants

[2] Except for the *Session* event.

[3] Except for *Trust* primitives, which cannot be instantiated into events because they are global assumptions.

[4] Computations can involve different values of the same variables from different sessions.

can take. The set $Val_\perp$ is defined as $Val \cup \{\perp\}$ where $\perp \notin Val$ is a specific symbol used to denote that a variable or a constant has not been assigned yet.

The semantics of an architecture follows the approach introduced in [2]. Each component is associated with a state. Each event in a trace of events affects the state of each component involved by the event. The semantics of an architecture is defined as the set of states reachable by compatible traces.

The state of a component is either the $Error$ state or a pair consisting of: (i) a variable state assigning values to variables, and (ii) a property state defining what is known by a component.

$$State_\perp = (State_V \times State_P) \cup \{Error\}$$
$$State_V = Var \cup Const \to \mathsf{List}(Val_\perp)$$
$$State_P = \{Eq\} \cup \{Trust_{i,j}\}$$

The data structure List over a set S denotes the finite ordered lists of elements of S, $\mathsf{size}(L)$ denotes the size of the list L, and $()$ is the empty list. For a non-empty list $L = (e_1, \ldots, e_n) \in S^n$ where $\mathsf{size}(L) = n \geq 1$, $L[m]$ denotes the element e_m for $1 \leq m \leq n$, $\mathsf{last}(L)$ denotes $L[n]$, and $\mathsf{append}(L, e)$ denotes the list $(e_1, \ldots, e_n, e) \in S^{n+1}$. Let $\sigma := (\sigma_1, \ldots, \sigma_N)$ denote the global state (*i.e.* the list of states of all components) defined over $(State_\perp)^N$ and σ_i^v and σ_i^{pk} denote, respectively, the variable and the knowledge state of the component C_i.

The variable state assigns values to variables and to constants (each constant is either undefined or taking a single value). $\sigma_i^v(X)[m]$ (resp. $\sigma_i^v(c)[m]$) denotes the m-th entry of the variable state of $X \in Var$ (resp. $c \in Const$). The initial state of an architecture A is denoted by $Init^A = \langle Init_1^A, \ldots, Init_N^A \rangle$ where: $\forall C_i$: $Init_i^A = (Empty, \{Trust_{i,j} \mid \exists C_j\text{: } Trust_{i,j} \in A\})$. $Empty$ associates to each variable and constant a list made of a single undefined value ($\perp$). We assume that, in the initial state, the system is in its first session. Alternatively, we could set empty lists in the initial state and assume that every consistent trace begins with a $Session$ event.

Let $S_T : Trace \times (State_\perp)^N \to (State_\perp)^N$ and $S_E : Event \times (State_\perp)^N \to (State_\perp)^N$ be the following two functions. S_T is defined recursively by iteration of S_E: for all state $\sigma \in (State_\perp)^N$, event $\epsilon \in Event$ and consistent trace $\theta \in Trace$, $S_T(\langle\rangle, \sigma) = \sigma$ and $S_T(\epsilon \cdot \theta, \sigma) = S_T(\theta, S_E(\epsilon, \sigma))$. The modification of a state is noted $\sigma[\sigma_i/(v, pk)]$ the variable and knowledge states of C_i are replaced by v and pk respectively. $\sigma[\sigma_i/Error]$ denotes that the $Error$ state is reached for component C_i. We assume that a component reaching an $Error$ state no longer gets involved in any later action (until a reset of the system). The function S_E is defined event per event.

The effect of $Has_i(X : V)$ and $Receive_{i,j}(S, \{(X : V)\})$ on the variable state of component C_i is the replacement of the last value of the variable X by the value V: $\mathsf{last}(\sigma_i^v(X)) := V$. This effect is denoted by $\sigma_i^v[X/V]$.

For instance, we have

$$S_E(Has_i(X : V), \sigma) = \sigma[\sigma_i/(\sigma_i^v[X/V], \sigma_i^{pk})]$$

where $\sigma_i^v[X/V]$ means that new values V replace the values of the variables X, and σ_i^{pk} stands for the property component of the state of C_i.

Similarly, one can write

$$S_E(Receive_{i,j}(S, \{X : V\}), \sigma) = \sigma[\sigma_i/(\sigma_i^v[X/V], \sigma_i^{pk})]$$

In the case of constants, the value V is determined by the interpretation of c (as in the function symbols in the computation).

The effect of $Compute_G(X = T)$ is to assign to X, for each component $C_i \in G$, the value V produces by the evaluation (denoted ε) of T. The new knowledge is the equation $X = T$. A computation may involve values of variables from different sessions. As a result, some consistency conditions must be met, otherwise an error state is reached:

$$S_E(Compute_G(X = T), \sigma) = \begin{cases} \sigma[\forall C_i \in G : \sigma_i/(\sigma_i^v[X/V], \sigma_i^{pk} \cup \{X = T\})] \\ \qquad \text{if the condition on the computation holds,} \\ \sigma[\sigma_i/Error] \text{ otherwise,} \end{cases}$$

where $V := \varepsilon\,(T, \cup_{C_i \in G} \sigma_i^v)$. For each $\tilde{X}^{(n)} \in T$, the evaluation of T is done with respect to the n last values of $\tilde{X}$ that are fully defined. An error state is reached if n such values are not available. The condition on the computation is then: $\forall C_i \in G, \tilde{X}^{(n)} \in T$: $\mathsf{size}\big(\{m \mid \sigma_i^v\big(V\big(\tilde{X}\big)\big)[m] \text{ is fully defined}\}\big) \geq n$.

Semantics of the verification events are defined according to the (implicit) semantics of the underlying verification procedures. In each case, the knowledge state of the component is updated if the verification passes, otherwise the component reaches an $Error$ state. The variable state is not affected.

$$S_E(Verify_i(Proof_j(E)), \sigma) = \begin{cases} \sigma[\sigma_i/(\sigma_i^v, \sigma_i^{pk} \cup new_{Proof}^{pk})] \\ \qquad \text{if the proof is valid,} \\ \sigma[\sigma_i/Error] \text{ otherwise,} \end{cases}$$

$$S_E(Verify_i(Attest_j(E)), \sigma) = \begin{cases} \sigma[\sigma_i/(\sigma_i^v, \sigma_i^{pk} \cup new_{Attest}^{pk})] \\ \qquad \text{if the attestation is valid,} \\ \sigma[\sigma_i/Error] \text{ otherwise.} \end{cases}$$

The new knowledge new_{Proof}^{pk} and new_{Attest}^{pk} are defined as:

$$new_{Proof}^{pk} := \left\{ Eq \;\middle|\; Eq \in E \;\vee\; \begin{pmatrix} \exists C_k : Attest_k(E') \in E \;\wedge\; Eq \in E' \\ \wedge\; Trust_{i,k} \in \sigma_i^{pk} \end{pmatrix} \right\} \text{ and}$$

$$new_{Attest}^{pk} := \{Eq \mid Eq \in E \;\wedge\; Trust_{i,j} \in \sigma_i^{pk}\}.$$

In the session case, the knowledge state is reinitialized and a new entry is added in the variable states:

$$S_E(Session, \sigma) = \sigma[\forall i : \sigma_i/(upd^v, \{Trust_{i,j} \mid \exists C_j : Trust_{i,j} \in A\})],$$

where the new variable state upd^v is such that $\sigma_i^v(X) := \mathsf{append}(\sigma_i^v(X), \bot)$ for all variables $X \in Var$, and $\sigma_i^v(c) := \mathsf{append}(\sigma_i^v(c), \mathsf{last}(\sigma_i^v(c)))$ for all constants

$c \in Const$. The session event is not local to a component, all component states are updated. As a result, we associate to each global state σ a unique number, noted $\mathsf{s}(\sigma)$, which indicates the number of sessions. In the initial state, $\mathsf{s}(\sigma) := 1$, and at each $Session$ event, $\mathsf{s}(\sigma)$ is incremented.

In the reset case, all values are dropped and the initial state is restored: $S_E(Reset, \sigma) = Init^A$.

This ends the definition of the semantics of trace of events. The semantics $S(A)$ of an architecture A is defined as the set of states reachable by compatible traces.

5.2 Extension of the privacy logic

The privacy logic is enhanced to express access to n values of a given variable. The formula Has_i represents $n \geq 1$ accesses by C_i to some variable X.

$$\varphi ::= Has_i(X^{(n)}) \mid Has_i(c) \mid Has_i^{none}(X) \mid Has_i^{none}(c) \mid K_i(Eq) \mid \varphi_1 \wedge \varphi_2$$
$$Eq ::= Pred(T_1, \ldots, T_m)$$

Several values of the same variables from different sessions can provide information about other variables, which is expressed through the dependence relation.

The semantics $S(\varphi)$ of a property $\varphi \in \mathcal{L}_P$ remains defined as the set of architectures where φ is satisfied. The fact that φ is satisfied by a (consistent) architecture A is defined as follows.

- A satisfies $Has_i(X^{(n)})$ if there is a reachable state in which X is fully defined (at least) $n \geq 1$ times.
- A satisfies $Has_i(c)$ if there is a reachable state in which c is fully defined.
- A satisfies $Has_i^{none}(X)$ (resp. $Has_i^{none}(c)$) if no compatible trace leads to a state in which C_i assigns a value to X (resp. c).
- A satisfies $K_i(Eq)$ if for all reachable states, there exists a state in the same session in which C_i can derive Eq.
- A satisfies $\varphi_1 \wedge \varphi_2$ if A satisfies φ_1 and A satisfies φ_2.

A set of deductive rules for this privacy logic is given in Figure 6. One can show that this axiomatics is sound and complete with respect to the semantics above. The soundness theorem states that for all A, if $A \vdash \varphi$, then $A \in S(\varphi)$. Completeness means that for all A, if $A \in S(\varphi)$ then $A \vdash \varphi$.

Due to the length of the proofs and the lack of place, we only give sketch for these proofs. Soundness is proved by induction on the derivation tree. For each theorem $A \vdash \varphi$, one can find traces satisfying the claimed property, or show that all traces satisfy the claimed property (depending on the kind of property). Completeness is shown by induction on the property φ. For each property belonging to the semantics, one can exhibit a tree that derives it from the architecture.

A trace is said to be a covering trace if it contains an event corresponding to each primitive specified in an architecture A (except trust relations) and if for each primitive it contains as much events as the multiplicity $^{(n)}$ of the primitive.

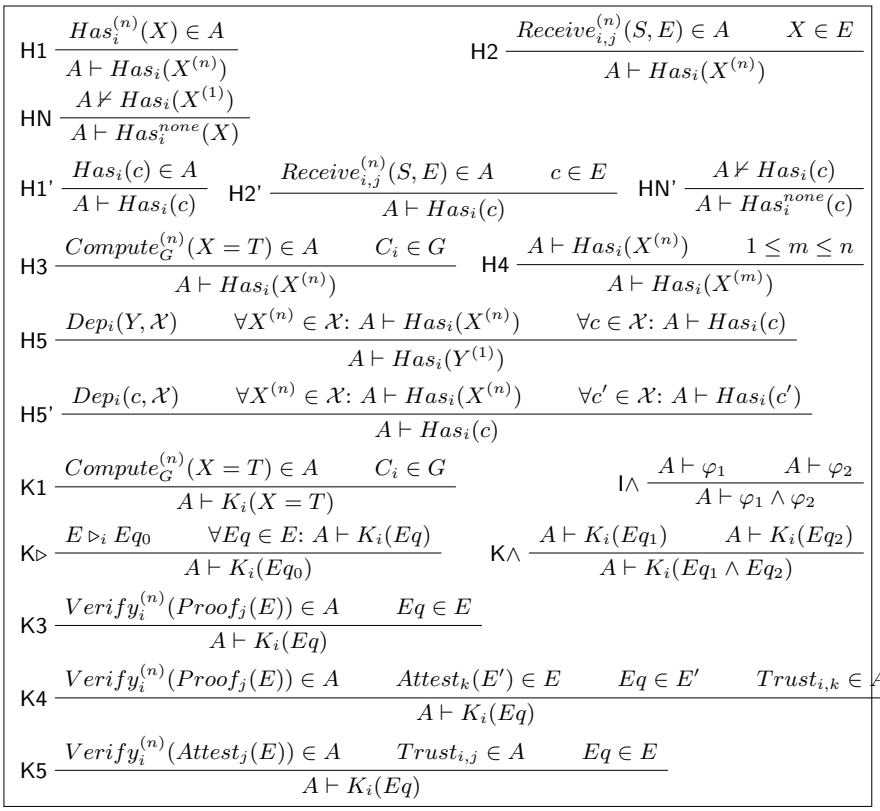


Fig. 6. Set of deductive rules for the extended privacy logic

As a first step to prove soundness, it is shown that for all consistent architecture A, there exists a consistent trace $\theta \in T(A)$ that covers A.

Then the soundness is shown by induction on the depth of the tree $A \vdash \varphi$.

- Let us assume that $A \vdash Has_i(X^{(n)})$, and that the derivation tree is of depth 1. By definition of $\mathcal{D}$, such a proof is obtained by application of (H1), (H2) or (H3). In each case, it is shown (thanks to the existence of covering traces) that an appropriate trace can be found in the semantics of A, hence $A \in S(Has_i(X^{(n)}))$. The case of $A \vdash Has_i(c)$ is very similar.
- Let us assume that $A \vdash K_i(Eq)$, and that the derivation tree is of depth 1. By definition of $\mathcal{D}$, such a proof is obtained by application of (K1), (K2), (K3), (K4) or (K5). In each case, starting from a state $\sigma' \in S_i(A)$ such that $\mathsf{s}(\sigma') \geq n$, it is first shown that there exists a covering trace $\theta \geq \theta'$ that extends θ' and that contains n corresponding events $Compute_G(X = T) \in \theta$ in n distinct sessions (for the K1 case, and other events for the other rules).

Then by the properties of the deductive algorithmic knowledge, it is shown that the semantics of the property $A \in S(K_i(X = T))$ holds.

- Let us assume that $A \vdash Has_i(X^{(n)})$, and that the derivation tree is of depth strictly greater than 1. By definition of $\mathcal{D}$, such a proof is obtained by application of (H4) or (H5).

  In the first case, by the induction hypothesis and the semantics of properties, there exists a reachable state $\sigma \in S(A)$ and n indices $i_1, \ldots, i_n$ such that $\sigma_i^v(X)[i_l]$ is fully defined for all $l \in [1, n]$. This gives, *a fortiori*, $A \in S(Has_i(X^{(m)}))$ for all m such that $1 \leq m \leq n$.

  In the second case, we have that $(Y, \{X_1^{(n_1)}, \ldots, X_m^{(n_m)}, c_1, \ldots, c_q\}) \in Dep_i$, that $\forall l \in [1, m] : A \vdash Has_i(X_l^{(n_l)})$ and $\forall l \in [1, q] : A \vdash Has_i(c_l)$. The proof shows the existence of a covering trace that contains an event $Compute_G$ $(Y = T)$ (where $i \in G$), allowing to conclude that $A \in S(Has_i(Y^{(1)}))$.

  Again, the corresponding cases for constant are very similar.
- A derivation for Has^{none} is obtained by application of (HN). The proof assume, towards a contradiction, that $A \notin S(Has_i^{none}(X))$. It is shown, by the architecture semantics, that there exists a compatible trace that enable to derive $A \vdash Has_i^{(1)}(X)$. However, since (HN) was applied, we have $A \nvdash Has_i^{(1)}(X)$, hence a contradiction.
- The last case (the conjunction $\wedge$) is fairly straightforward.

The completeness is proved by induction over the definition of φ.

- Let us assume that $A \in S(Has_i(X^{(n)}))$. By the architecture semantics and the semantics of traces, it is shown that the corresponding traces either contain events where X is computed, received or measured, or that some dependence relation on X exists. In the first case, we have $A \vdash Has_i(X^{(n)})$ by applying (respectively) (H1), (H2), or (H3) (after an eventual application of (H4)). In the last case, the proof shows how to exhibit a derivation tree to obtain $A \vdash Has_i(X^{(n)})$ (the (H5) rule is used).
- Let us assume that $A \in S(Has_i^{none}(X))$. By the semantics of properties, this means that in all reachable states, X does not receive any value. The proof shows that $A \nvdash S(Has_i(X^{(1)}))$, otherwise $A \in S(Has_i^{none}(X))$ would be contradicted. So as a conclusion, $A \vdash Has_i^{none}(X)$ by applying (HN).
- The constant cases $A \in S(Has_i(c)$ and $A \in S(Has_i^{none}(c))$ case are similar to the variable cases.
- Let us assume that $A \in S(K_i(Eq))$. By the semantics of properties this means that for all reachable states, there exists a later state in the same session where the knowledge state enables to derive Eq. By the semantics of architecture, we can exhibit a compatible trace that reaches a state where Eq can be derived. By the semantics of compatible traces, the proof shows, by reasoning on the events on the traces, that $A \vdash K_i(Eq)$ by applying either (K1), (K2), (K3), (K4) or (K5).
- Finally the conjunctive case is straightforward.

6 Extension of the Match-On-Card to the identification paradigm

We now show of the extended framework can be used to reason about the privacy properties of a biometric system where some information leaks after several sessions of the same protocol.

The biometric system introduced in [9] aims at extending the MOC technology (cf. Section 4.3) to the identification paradigm. A quantized version – corresponding to short binary representations of the templates – of the database is stored inside a secure module, playing the role of the card in the MOC case. From each biometric reference template, a quantization is computed, using typically a secure sketch scheme [24,14]. The reference database is encrypted and stored outside the secure module, whereas the quantizations of the templates are stored inside.

The verification step is processed as follows. Suppose one wants to identify himself in the system. A terminal captures the fresh biometrics, extracts a template, computes its quantization `qs` and sends them to the secure module. Then, the module proceeds to a comparison between the fresh quantization and all enrolled quantizations `qr`. The C nearest quantizations, for some parameter C of the system, are the C potential candidates for the identification. Then, the module queries the C corresponding (encrypted) templates to the database (by using the list of indices `ind` of those C nearest quantized versions `qr` of the enrolled templates). This gives the module the access to the set `sebr` of the C encrypted templates. The module decrypts them, and compares them with the fresh template `bs`. The module finally sends its response to the terminal: 1 if one of the enrolled templates is close enough to the fresh template, 0 otherwise. Figure 7 gives a graphical representation of the resulting architecture.

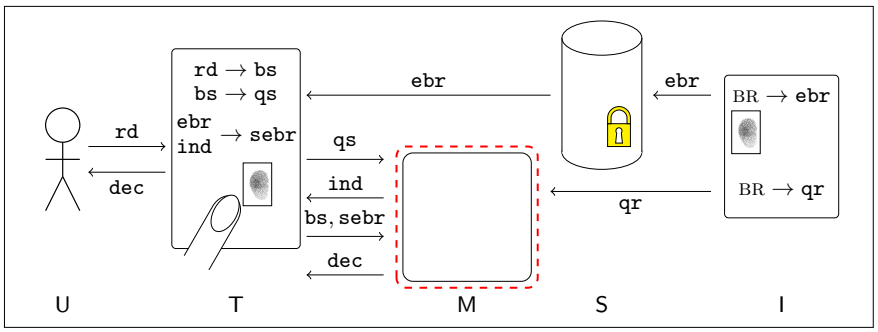


Fig. 7. Architecture of the extension of the Match-On-Card technology to biometric identification. The dotted red line indicates the location of the comparison.

N denotes the size of the database (*i.e.* the number of enrolled users), Q the size of the quantizations, and C the number of indices asked by the card. The ranges are $Range(\text{BR}, \texttt{ebr}, \texttt{qr}) = \text{N}$, $Range(\texttt{rd}, \text{THR}, \texttt{bs}, \texttt{qs}, \texttt{dec}) = 1$, and $Range(\texttt{ind}, \texttt{sebr}, \texttt{sbr}) = \text{C}$. The set Fun of functions contains the extraction procedure $Extract$, the encryption and decryption procedures Enc and Dec, the (non-invertible) quantization $Quant$ of the biometric templates, the comparison of the quantizations $QComp$, which takes as inputs two sets of quantizations and the parameter C, the selection of the encrypted templates $EGet$, and finally the matching μ, which takes as arguments two biometric templates and the threshold THR.

The biometric reference templates are enrolled by the issuer ($Has_{\mathsf{I}}(\text{BR})$). A verification process is initiated by the terminal T receiving as input a raw biometric data $\texttt{rd}$ from the user U. T extracts the fresh biometric template $\texttt{bs}$ from $\texttt{rd}$ using the function $Extract \in Fun$. The architecture then contains, as other biometric systems, $Receive_{\mathsf{T},\mathsf{U}}(\{\}, \{\texttt{rd}\})$ and $Compute_{\mathsf{T}}(\texttt{bs} = Extract(\texttt{rd}))$ and the Dep_{T} relation is such that $(\texttt{bs}, \{\texttt{rd}\}) \in Dep_{\mathsf{T}}$. The user receives the final decision $\texttt{dec}$ from the terminal: $Receive_{\mathsf{U},\mathsf{T}}(\{\}, \{\texttt{dec}\})$. To sum up, the architecture is described as follows in the framework of Section 2:

$$
\begin{aligned}
A^{\mathsf{mi}} := \big\{ & Has_{\mathsf{I}}(\text{BR}), Has_{\mathsf{U}}(\texttt{rd}), Has_{\mathsf{M}}(\text{C}), Has_{\mathsf{M}}(\text{THR}),\\
& Compute_{\mathsf{I}}(\texttt{ebr} = Enc(\text{BR})), Compute_{\mathsf{I}}(\texttt{qr} = Quant(\text{BR})),\\
& Compute_{\mathsf{T}}(\texttt{bs} = Extract(\texttt{rd})), Compute_{\mathsf{T}}(\texttt{sebr} = EGet(\texttt{ebr}, \texttt{ind})),\\
& Compute_{\mathsf{T}}(\texttt{qs} = Quant(\texttt{bs})), Compute_{\mathsf{M}}(\texttt{ind} = QComp(\texttt{qs}, \texttt{qr}, \text{C})),\\
& Compute_{\mathsf{M}}(\texttt{sbr} = Dec(\texttt{sebr})), Compute_{\mathsf{M}}(\texttt{dec} = \mu(\texttt{sbr}, \texttt{bs}, \text{THR})),\\
& Receive_{\mathsf{S},\mathsf{I}}(\{Attest_{\mathsf{I}}(\texttt{ebr} = Enc(\text{BR}))\}, \{\texttt{ebr}\}), Receive_{\mathsf{T},\mathsf{U}}(\{\}, \{\texttt{rd}\}),\\
& Receive_{\mathsf{T},\mathsf{S}}(\{Attest_{\mathsf{I}}(\texttt{ebr} = Enc(\text{BR}))\}, \{\texttt{ebr}\}), Receive_{\mathsf{M},\mathsf{T}}(\{\}, \{\texttt{qs}\}),\\
& Receive_{\mathsf{M},\mathsf{I}}(\{Attest_{\mathsf{I}}(\texttt{qr} = Quant(\text{BR}))\}, \{\texttt{qr}\}), Receive_{\mathsf{T},\mathsf{M}}(\{\}, \{\texttt{ind}\}),\\
& Receive_{\mathsf{M},\mathsf{T}}(\{\}, \{\texttt{sebr}, \texttt{bs}\}), Receive_{\mathsf{T},\mathsf{M}}(\{\}, \{\texttt{dec}\}),\\
& Trust_{\mathsf{T},\mathsf{I}}, Trust_{\mathsf{M},\mathsf{I}}, Trust_{\mathsf{T},\mathsf{M}}, Verify_{\mathsf{T}}(Attest_{\mathsf{I}}(\texttt{ebr} = Enc(\text{BR}))),\\
& Verify_{\mathsf{T}}(\{Attest_{\mathsf{M}}(\texttt{dec} = \mu(\texttt{sbr}, \texttt{bs}, \text{THR}))\}),\\
& Verify_{\mathsf{M}}(Attest_{\mathsf{I}}(\texttt{qr} = Quant(\text{BR}))), Verify_{\mathsf{T}}(\{Attest_{\mathsf{M}}(\texttt{sbr} = Dec(\texttt{ebr}))\})\big\}
\end{aligned}
$$

The issuer encrypts the templates and computes the quantizations, which is expressed by the dependencies: $Dep_{\mathsf{I}}^{\mathsf{mi}} := \{(\texttt{ebr}, \{\text{BR}\}), (\texttt{qr}, \{\text{BR}\})\}$. The terminal and module computations are reflected in the dependencies as well: $Dep_{\mathsf{T}}^{\mathsf{mi}} := \{(\texttt{bs}, \{\texttt{rd}\}), (\texttt{qs}, \{\texttt{bs}\})\}, (\texttt{sebr}, \{\texttt{bs}, \texttt{ind}\})\}$. The dependency relation of the module reflects its ability to decrypt the templates: $Dep_{\mathsf{M}}^{\mathsf{mi}} := \{(\texttt{ind}, \{\texttt{qs}, \texttt{qr}, \text{C}\}), (\texttt{sbr}, \{\texttt{sebr}\}), (\texttt{dec}, \{\texttt{sbr}, \texttt{bs}, \text{THR}\}), (\text{BR}, \{\texttt{ebr}\})\}$. The absence of such a relation in other dependencies prevents the corresponding components to get access to the plain references, even if they get access to the ciphertexts.

6.1 Learning from the selected quantizations

Let us now discuss the following point: the formalism of Section 2 is insufficient to consider the leakage of the sensitive biometric data stored inside the module. In A^{mi}, we would like that the terminal gets no access to the quantizations: $A^{\mathsf{mi}} \in Has^{none}_{\mathsf{T}}(\mathtt{qr})$. It is indeed possible to derive $A^{\mathsf{mi}} \vdash Has^{none}_{\mathsf{T}}(\mathtt{qr})$, thanks to the ($\mathsf{HN}$) rule. According to the notations of [2], where $Has_i(X)$ stands for $Has_i(X^{(1)})$ in this paper, we have:

$$\mathsf{HN}\ \frac{\begin{array}{cc} \nexists X : Dep_{\mathsf{T}}(\mathtt{qr}, X) \in A^{\mathsf{mi}} & \nexists j, S : Receive_{\mathsf{T},j}(S, \{\mathtt{qr}\}) \in A^{\mathsf{mi}} \\ Has_{\mathsf{T}}(\mathtt{qr}) \notin A^{\mathsf{mi}} & \nexists T : Compute_{\mathsf{T}}(\mathtt{qr} = T) \in A^{\mathsf{mi}} \end{array}}{A \nvdash Has_{\mathsf{T}}(\mathtt{qr})} \over A \vdash Has^{none}_{\mathsf{T}}(\mathtt{qr})$$

This corresponds to the intuition saying that quantizations are protected since they are stored in a secure hardware element.

However, an attack (described in [12]) shows that, in practice, quantizations can be learned if a sufficient number of queries to the module is allowed. The attack roughly proceeds as follows (we drop the masks for sake of clarity). The attacker maintains a $\mathrm{N} \times \mathrm{Q}$ table (say T) of counters for each bit to be guessed. All entries are initialized to 0. Then it picks Q-bits random vector Q and sends it to the module. The attacker observes the set of indices $\mathtt{ind} \subseteq [1, \mathrm{N}]$ corresponding to the encrypted templates asked by the module. It updates its table T as follows, according to its query Q and the response $\mathtt{ind}$: for each $i \in [1, \mathrm{N}]$ and $j \in [1, \mathrm{Q}]$, it decrements the entry $T[i][j]$ if $Q[j] = 0$, and increments it if $Q[j] = 1$. At the end of the attack, the N quantizations are guessed from the signs of the counters.

The number of queries made to the module is the crucial point in the attack above (and generally in other black-box attacks against biometric systems [12]). Our extended model enables to introduce a bound on the number of actions allowed to be performed. We now use this model to integrate such a bound in the formal architecture description. Let $A^{\mathsf{mi\text{-}e}}(n)$ be the following architecture, for some $n \geq 1$:

$$\begin{aligned} A^{\mathsf{mi\text{-}e}}(n) := \big\{ & Has_{\mathsf{I}}(\mathrm{BR}), Has^{(n)}_{\mathsf{U}}(\mathtt{rd}), Has_{\mathsf{M}}(\mathrm{C}), Has_{\mathsf{M}}(\mathrm{THR}), \\ & Compute^{(n)}_{\mathsf{I}}(\mathtt{ebr} = Enc(\mathrm{BR})), Compute^{(n)}_{\mathsf{I}}(\mathtt{qr} = Quant(\mathrm{BR})), \\ & Compute^{(n)}_{\mathsf{T}}(\mathtt{bs} = Extract(\mathtt{rd})), Compute^{(n)}_{\mathsf{T}}(\mathtt{sebr} = EGet(\mathtt{ebr}, \mathtt{ind})), \\ & Compute^{(n)}_{\mathsf{T}}(\mathtt{qs} = Quant(\mathtt{bs})), Compute^{(n)}_{\mathsf{M}}(\mathtt{ind} = QComp(\mathtt{qs}, \mathtt{qr}, \mathrm{C})), \\ & Compute^{(n)}_{\mathsf{M}}(\mathtt{sbr} = Dec(\mathtt{sebr})), Compute^{(n)}_{\mathsf{M}}(\mathtt{dec} = \mu(\mathtt{sbr}, \mathtt{bs}, \mathrm{THR})), \\ & Receive^{(n)}_{\mathsf{S},\mathsf{I}}(\{Attest_{\mathsf{I}}(\mathtt{ebr} = Enc(\mathrm{BR}))\}, \{\mathtt{ebr}\}), Receive^{(n)}_{\mathsf{T},\mathsf{U}}(\{\}, \{\mathtt{rd}\}), \\ & Receive^{(n)}_{\mathsf{T},\mathsf{S}}(\{Attest_{\mathsf{I}}(\mathtt{ebr} = Enc(\mathrm{BR}))\}, \{\mathtt{ebr}\}), Receive^{(n)}_{\mathsf{M},\mathsf{T}}(\{\}, \{\mathtt{qs}\}), \\ & Receive^{(n)}_{\mathsf{M},\mathsf{I}}(\{Attest_{\mathsf{I}}(\mathtt{qr} = Quant(\mathrm{BR}))\}, \{\mathtt{qr}\}), Receive^{(n)}_{\mathsf{T},\mathsf{M}}(\{\}, \{\mathtt{ind}\}), \\ & Receive^{(n)}_{\mathsf{M},\mathsf{T}}(\{\}, \{\mathtt{sebr}, \mathtt{bs}\}), Receive^{(n)}_{\mathsf{T},\mathsf{M}}(\{\}, \{\mathtt{dec}\}), \end{aligned}$$

$$Trust_{\mathsf{T},\mathsf{I}}, Trust_{\mathsf{M},\mathsf{I}}, Trust_{\mathsf{T},\mathsf{M}}, Verify_{\mathsf{T}}^{(n)}(Attest_{\mathsf{I}}(\mathtt{ebr} = Enc(\text{BR}))),$$
$$Verify_{\mathsf{T}}^{(n)}(\{Attest_{\mathsf{M}}(\mathtt{dec} = \mu(\mathtt{sbr}, \mathtt{bs}, \text{THR}))\}),$$
$$Verify_{\mathsf{M}}^{(n)}(Attest_{\mathsf{I}}(\mathtt{qr} = Quant(\text{BR}))),$$
$$Verify_{\mathsf{T}}^{(n)}(\{Attest_{\mathsf{M}}(\mathtt{sbr} = Dec(\mathtt{ebr}))\})\}$$

In addition to the dependence of A^{mi}, the dependence relations indicates that the leakage is conditioned by a specific link mapping between the outsourced ciphertexts and the stored quantizations: $Dep_{\mathsf{T}}^{\mathsf{mi\text{-}e}}(\mathtt{qr}, \{\mathtt{ind}^{(\text{N}\cdot\text{Q})}, \mathtt{qs}^{(\text{N}\cdot\text{Q})}\})$. Furthermore, the module may learn the entire database $\mathtt{ebr}$ in a number of queries depending on the size of the database and the number of indices asked by the module: $Dep_{\mathsf{M}}^{\mathsf{mi\text{-}e}}(\mathtt{ebr}, \{\mathtt{sebr}^{(\lceil \text{N}/\text{C} \rceil)}\})$.

6.2 Strengthened variants of the architecture

Now, based on some counter-measures of the attacks indicated in [12], we express several variants of the architecture $A^{\mathsf{mi\text{-}e}}$. For each variant, the deductive rules $\mathcal{D}$ for the property language $\mathcal{L}_P$ are used to show that, for some conditions on the parameters, the quantizations $\mathtt{qr}$ are protected.

Variant 1 As a first counter-measure, the module could ask the entire database at each invocation. It is rather inefficient, and, in some sense, runs against to initial motivation of its design. However, this can be described within the language $\mathcal{L}_A$, and, in practice, can be manageable for small databases. This architecture, denoted $A^{\mathsf{mi\text{-}e1}}$, is given by $A^{\mathsf{mi\text{-}e}}(n)$ for some $n \geq 1$, except that $Dep_{\mathsf{T}}^{\mathsf{mi\text{-}e1}} := Dep_{\mathsf{T}}^{\mathsf{mi}}$. It is now possible to prove that the quantizations are protected, even in presence of several executions of the protocols. Since the relations Dep_{T} no longer contains a dependence leading to $\mathtt{qr}$, an application of (HN) becomes possible and gives the expected property.

$$\frac{\nexists X : Dep_{\mathsf{T}}(\mathtt{qr}, X) \in A^{\mathsf{mi\text{-}e1}} \qquad \nexists j : Receive_{\mathsf{T},j}^{(n)}(S, \{\mathtt{qr}\}) \in A^{\mathsf{mi\text{-}e1}} \qquad Has_{\mathsf{T}}^{(n)}(\mathtt{qr}) \notin A^{\mathsf{mi\text{-}e1}} \qquad \nexists T : Compute_{\mathsf{T}}^{(n)}(\mathtt{qr} = T) \in A^{\mathsf{mi\text{-}e1}}}{\mathsf{HN}\ \dfrac{\forall n : A \nvdash Has_{\mathsf{T}}(\mathtt{qr}^{(n)})}{A \vdash Has_{\mathsf{T}}^{none}(\mathtt{qr})}}$$

Variant 2 In the precedent variant, the effect of the counter-measure is the withdrawal of the dependence relation. We now consider architectures where such a dependency is still given, but where counter-measures are used to prevent a critical bound on the number of queries to be reached.

A first measure is to block the number of attempts the terminal can make. The module can detect it and refuse to respond. This architecture, denoted $A^{\mathsf{mi\text{-}e2}}$, is given by $A^{\mathsf{mi\text{-}e}}(\text{B})$, for some $\text{B} \ll \text{N}\cdot\text{Q}$. As a result, the $Has_i^{none}(\mathtt{qr})$ property can be derived. In particular one must show that $A^{\mathsf{mi\text{-}e2}} \nvdash Has_{\mathsf{T}}(\mathtt{ind}^{(\text{N}\cdot\text{Q})})$, in order to prevent the dependence rule H5 to be applied.

$$\frac{\begin{array}{cc} & Has_{\mathsf{T}}^{(\mathrm{B})}(\mathtt{ind}) \in A^{\mathsf{mi\text{-}e2}} \qquad \mathrm{B} < \mathrm{N} \cdot \mathrm{Q} \\ \nexists S : Receive_{\mathsf{T},\mathsf{M}}^{(\mathrm{B})}(S, \{\mathtt{ind}\}) \in A^{\mathsf{mi\text{-}e2}} & \nexists T : Compute_{\mathsf{T}}^{(\mathrm{B})}(\mathtt{ind} = T) \in A^{\mathsf{mi\text{-}e2}} \end{array}}{A^{\mathsf{mi\text{-}e2}} \nvdash Has_{\mathsf{T}}(\mathtt{ind}^{(\mathrm{N}\cdot\mathrm{Q})})}$$

An application of HN enables to conclude.

$$\mathsf{HN}\,\frac{\begin{array}{cc} & Has_{\mathsf{T}}^{(\mathrm{B})}(\mathtt{qr}) \notin A^{\mathsf{mi\text{-}e2}} \\ Dep_{\mathsf{T}}^{\mathsf{mi\text{-}e2}}(\mathtt{qr}, \{\mathtt{ind}^{(\mathrm{N}\cdot\mathrm{Q})}\}) \in A^{\mathsf{mi\text{-}e2}} & \nexists j : Receive_{\mathsf{T},j}^{(\mathrm{B})}(S, \{\mathtt{qr}\}) \in A^{\mathsf{mi\text{-}e2}} \\ A^{\mathsf{mi\text{-}e2}} \nvdash Has_{\mathsf{T}}(\mathtt{ind}^{(\mathrm{N}\cdot\mathrm{Q})}) & \nexists T : Compute_{\mathsf{T}}^{(\mathrm{B})}(\mathtt{qr} = T) \in A^{\mathsf{mi\text{-}e2}} \end{array}}{A^{\mathsf{mi\text{-}e2}} \vdash Has_{\mathsf{T}}^{none}(\mathtt{qr})}\,\frac{A^{\mathsf{mi\text{-}e2}} \nvdash Has_{\mathsf{T}}(\mathtt{qr}^{(1)})}{}$$

Variant 3 In the precedent variant, the terminal cannot accumulate enough information since he cannot query the module enough times to derive a useful knowledge. We now describe a variant where the terminal has no bound on the number of times it asks the module, but where the system is regularly reinitialised, so that the accumulated information becomes useless.

The leakage of the system runtime is dependent on some association between the quantizations $\mathtt{qr}$ and the encrypted database $\mathtt{ebr}$; namely the association π that maps the quantization $\mathtt{qr}[i] = Quant(\mathrm{BR}[\pi(i)])$ to the encrypted template from which it has been computed $\mathtt{ebr}[\pi(i)] = Enc(\mathrm{BR}[\pi(i)])$. Once this mapping is changed, the information is cancelled. For instance the database can be randomly permuted after B queries to the secure module.

Formally, this is caught by adding a *Reset* primitive to the architecture. Let $A^{\mathsf{mi\text{-}e3}}$ be the architecture defined as $A^{\mathsf{mi\text{-}e3}} := A^{\mathsf{mi\text{-}e2}} \cup \{Reset\}$. The semantics of the *Reset* events ensures that no more than B values of $\mathtt{ind}$ will be gathered by the terminal for a fixed mapping. The proof that $A^{\mathsf{mi\text{-}e3}} \vdash Has_{\mathsf{T}}^{none}(\mathtt{qr})$ is as the proof that $A^{\mathsf{mi\text{-}e2}} \vdash Has_{\mathsf{T}}^{none}(\mathtt{qr})$.

7 Related works

Privacy concerns related to the use of biometric data has attracted a lot of attention in the media (for instance, with the introduction of a fingerprint identity sensor in iPhones) and among lawyers and policy makers[5]. Most studies in the computer science community are done on a case by case basis and at a lower level than the architectures described here. For instance, [43] proposes a security model for biometric-based authentication taking into account privacy properties – including impersonation resilience, identity privacy or transaction anonymity – and applies it to biometric authentication. The underlying proofs rely on cryptographic techniques related to the ElGamal public key encryption scheme. Other

[5] For example with a proposal adopted by the French Senate in May 2014 to introduce stronger requirements for the use of biometrics.

works such as [25,28,29] develop formal models from an information theoretic perspective relying on specific representations of biometric templates akin to error correcting codes.

As far as formal approaches to privacy are concerned, two main categories can be identified: the qualitative approach and the quantitative approach. Most proposals of the first category rely on a language that can be used to define systems and to express privacy properties. For example process calculi such as the applied pi-calculus [1] have been applied to define privacy protocols [13]. Other studies [4,5] involve dedicated privacy languages. The main departure of the approach advocated in this paper with respect to this trend of work is that we reason at the level of architectures, providing ways to express properties without entering into the details of specific protocols. Proposals of the second category rely on privacy metrics such as k-anonymity, l-diversity, or ϵ-differential privacy [15] which can be seen as ways to measure the level of privacy provided by an algorithm. Methods [32] have been proposed to design algorithms achieving privacy metrics or to verify that a system achieves a given level of privacy. The contributions on privacy metrics are complementary to the work described in this paper. We follow a qualitative (or logical) approach here, proving that a given privacy property is met (or not) by an architecture. As suggested in the next section, an avenue for further research would be to cope with quantitative reasoning as well, using inference systems to derive properties expressed in terms of privacy metrics.

Several authors [20,26,33,34,41] have already pointed out the complexity of "privacy engineering" as well as the "richness of the data space"[20] calling for the development of more general and systematic methodologies for privacy by design. For example, [26,31] point out the complexity of the implementation of privacy and the large number of options that designers have to face. To address this issue and favour the adoption of these tools, [26] proposes a number of guidelines for the design of compilers for secure computation and zero-knowledge proofs whereas [17] provides a language and a compiler to perform computations on private data by synthesising zero-knowledge protocols. None of these proposals addresses the architectural level and makes it possible to get a global view of a system and to reason about its underlying trust assumption.

8 Conclusion

This work is the result of a collaboration between academics, industry and lawyers to show the applicability of the privacy by design approach to biometric systems and the benefit of formal methods to this end. Indeed, even if privacy by design becomes a legal obligation in the European Union [36] its application to real systems is far from obvious. We have presented in the same formal framework a variety of architectural options for privacy preserving biometric systems. We also have introduced an extension of this formal framework in order to catch the leakage due to the system runtime.

One of the main advantages of the approach is to provide formal justifications for the architectural choices and a rigorous basis for their comparison. Table 1 summarizes the main properties of the architectures reviewed in the first part of this paper. One of the most interesting pieces of information is the trust assumptions which are highlighted by the model. The first line shows that A_{ed} is the architecture in which the strongest trust in put in the terminal that does not have to trust any other component apart from the issuer and is able to get access to `br`. Architecture A_{hsm} is a variant of A_{ed}; it places less trust in the terminal that has to trust the hardware security module to perform the matching. A_{hom} is the architecture in which the terminal is less trusted: it has to trust the issuer, the hardware security module and the server for all sensitive operations and its role is limited to the collection of the fresh biometric trait and the computation of the fresh template. Architecture A_{moc} is similar to this respect but all sensitive operations are gathered into a single component, namely the smart card. It should be clear that no solution is inherently better than the others considering that extra, technical or non technical (organizational, economic, etc.) constraints may have to be taken into account and, depending on the context of deployment and the technology used, some trust assumptions may be more reasonable than others. From the strict privacy point of view however, the match on card architecture provides the best guarantees since only the secure module has to be trusted and this module hosts the matching operation and has exclusive access to the biometric reference template. In any event, the most important in the design process is to be able to understand the underlying trust assumptions of a particular choice of architecture and the consequences in terms of privacy.

Arch.	Computations	Template protection		Trust relations
	Location of the matching	Components accessing the references `br`	Components accessing the query `bs`	
A_{ed}	T	I, T	T	(T, I)
A_{hsm}	M	I, M	T, M	(T, I), (T, M)
A_{hom}	S	I	T	(T, I), (T, M), (T, S)
A_{moc}	M	M	T, M	(T, M)

Components are: user U, terminal T, server S, secure module M (used as a generic name for a hardware security module or a card C), issuer I.
A trust relation (i, j) means that component i trusts component j.

Table 1. Comparison between architectures

A benefit of the formal approach followed in this paper is that it can provide the foundations for a systematic approach to privacy by design. A proof of concept implementation of a system to support designers in their task has been proposed in [3]. In this system, the user can introduce his privacy and integrity

requirements (as well as any requirements imposed by the environment such as the location of a given operation on a designated component) and choose different options for the distribution of the operations and the trust assumptions. Architectures can be initially defined in a purely informal way and then translated into a formal model. A tool integrating the approach can be used by designers to build and verify architectures. Designers without any knowledge or even interest in formal methods can use the non formal part of the framework. They can explore the design space based on initial inputs provided in a non formal language and analyse the suggested architectures based on their graphical representations. Designers who want to obtain formal guarantees can try to prove properties of their architectures, either automatically or with the help of a verification tool integrated within the design environment.

As stated above, we focus on the architectural level in this paper. As a result, we do not cover the full development cycle. Preliminary work has been done to address the mapping from the architecture level to the protocol level to ensure that a given implementation, expressed as an applied pi-calculus protocol, is consistent with an architecture [42]. As far as the formal approach is concerned, it would also be interesting to study how it could be used in the context of future privacy certification schemes. This would be especially interesting in the context of the European General Data Protection Regulation [36] which promotes not only privacy by design but also privacy seals.

References

1. Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *ACM Symposium on Principles of Programming Languages – POPL'01*, pages 104–115. ACM Press, 2001.
2. Thibaud Antignac and Daniel Le Métayer. Privacy architectures: Reasoning about data minimisation and integrity. In *Security and Trust Management – STM'14*, volume 8743 of *LNCS*, pages 17–32. Springer, 2014.
3. Thibaud Antignac and Daniel Le Métayer. Trust driven strategies for privacy by design. In *Trust Management – IFIP-TM'15*, volume 454 of *IFIP*, pages 60–75. Springer, 2015.
4. Adam Barth, Anupam Datta, John C. Mitchell, and Helen Nissenbaum. Privacy and contextual integrity: Framework and applications. In *IEEE Symposium on Security and Privacy – S&P'06*, pages 184–198. IEEE Computer Society, 2006.
5. Moritz Y. Becker, Alexander Malkis, and Laurent Bussard. S4P: A generic language for specifying privacy preferences and policies. Technical report, Microsoft Research / IMDEA Software / EMIC, 2010.
6. BioPriv. Biometric systems Private by design. French ANR research project ANR-12-INSE-0013, 2013. `http://www.agence-nationale-recherche.fr/?Project=ANR-12-INSE-0013`.
7. Marina Blanton and Paolo Gasti. Secure and efficient protocols for iris and fingerprint identification. In *European Symposium on Research in Computer Security – ESORICS'11*, volume 6879 of *LNCS*, pages 190–209. Springer, 2011.
8. Julien Bringer, Hervé Chabanne, Malika Izabachène, David Pointcheval, Qiang Tang, and Sébastien Zimmer. An application of the Goldwasser–Micali cryptosys-

tem to biometric authentication. In *Australasian Conference on Information Security and Privacy – ACISP'07*, volume 4586 of *LNCS*, pages 96–106. Springer, 2007.
9. Julien Bringer, Hervé Chabanne, Tom A. M. Kevenaar, and Bruno Kindarji. Extending match–on–card to local biometric identification. In *Conference on Biometric ID Management and Multimodal Communication, BioID_MultiComm'09*, volume 5707 of *LNCS*, pages 178–186. Springer, 2009.
10. Julien Bringer, Hervé Chabanne, Daniel Le Métayer, and Roch Lescuyer. Privacy by design in practice: Reasoning about privacy properties of biometric system architectures. In *Formal Methods – FM'15*, volume 9109 of *LNCS*, pages 90–107. Springer, 2015.
11. Julien Bringer, Hervé Chabanne, Daniel Le Métayer, and Roch Lescuyer. Reasoning about privacy properties of biometric systems architectures in the presence of information leakage (Best Paper Award). In *Information Security Conference – ISC'15*, volume 9290 of *LNCS*, pages 493–510. Springer, 2015.
12. Julien Bringer, Hervé Chabanne, and Koen Simoens. Blackbox security of biometrics (invited paper). In *Conference on Intelligent Information Hiding and Multimedia Signal Processing – IIH-MSP'10*, pages 337–340. IEEE Computer Society, 2010.
13. Stéphanie Delaune, Steve Kremer, and Mark Ryan. Verifying privacy-type properties of electronic voting protocols: A taster. In *Towards Trustworthy Elections, New Directions in Electronic Voting*, volume 6000 of *LNCS*, pages 289–309. Springer, 2010.
14. Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *Advances in Cryptology – EUROCRYPT'04*, volume 3027 of *LNCS*, pages 523–540. Springer, 2004.
15. Cynthia Dwork. Differential privacy. In *International Colloquium on Automata, Languages and Programming – ICALP'06, Part II*, volume 4052 of *LNCS*, pages 1–12. Springer, 2006.
16. Ronald Fagin, Joseph Halpern, Yoram Moses, and Moshe Vardi. *Reasoning About Knowledge*. MIT Press, 2004.
17. Cédric Fournet, Markulf Kohlweiss, George Danezis, and Zhengqin Luo. ZQL: A compiler for privacy-preserving data processing. In *USENIX'13 Security Symposium*, pages 163–178. USENIX Association, 2013.
18. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *ACM Symposium on Theory of Computing – STOC'09*, pages 169–178. ACM Press, 2009.
19. Michelle Govan and Tom Buggy. A computationally efficient fingerprint matching algorithm for implementation on smartcards. In *Biometrics: Theory, Applications, and Systems – BTAS'07*, pages 1–6. IEEE, 2007.
20. Sada Gürses, Carmela Troncoso, and Claudia Díaz. Engineering Privacy by Design. Presented at the Computers, Privacy & Data Protection conference, 2011.
21. Joseph Y. Halpern and Riccardo Pucella. Dealing with logical omniscience. In *Conference on Theoretical Aspects of Rationality and Knowledge TARK'07*, pages 169–176, 2007.
22. Anil K. Jain, Arun Ross, and Salil Prabhakar. An introduction to biometric recognition. *IEEE Trans. Circuits Syst. Video Techn.*, 14(1):4–20, 2004.
23. Ari Juels and Madhu Sudan. A fuzzy vault scheme. *Des. Codes Cryptography*, 38(2):237–257, 2006.
24. Ari Juels and Martin Wattenberg. A fuzzy commitment scheme. In *ACM Conference on Computer and Communications Security – CCS'99*, pages 28–36. ACM Press, 1999.

25. Alper Kanak and Ibrahim Sogukpinar. BioPSTM: a formal model for privacy, security, and trust in template-protecting biometric authentication. *Security and Communication Networks*, 7(1):123–138, 2014.
26. Florian Kerschbaum. Privacy-preserving computation (position paper). In *Annual Privacy Forum on Privacy Technologies and Policy – APF'12*, volume 8319 of *LNCS*, pages 41–54. Springer, 2014.
27. Els Kindt. Best practices for privacy and data protection for the processing of biometric data. In Patrizio Campisi, editor, *Security and Privacy in Biometrics*, pages 339–367. Springer, 2013.
28. Lifeng Lai, Siu-Wai Ho, and H. Vincent Poor. Privacy-security trade-offs in biometric security systems – Part I: single use case. *IEEE Transactions on Information Forensics and Security*, 6(1):122–139, 2011.
29. Lifeng Lai, Siu-Wai Ho, and H. Vincent Poor. Privacy-security trade-offs in biometric security systems – Part II: multiple use case. *IEEE Transactions on Information Forensics and Security*, 6(1):140–151, 2011.
30. Huixian Li and Liaojun Pang. A novel biometric–based authentication scheme with privacy protection. In *Conference on Information Assurance and Security – IAS'09*, pages 295–298. IEEE Computer Society, 2009.
31. Matteo Maffei, Kim Pecina, and Manuel Reinert. Security and privacy by declarative design. In *IEEE Symposium on Computer Security Foundations – CSF'13*, pages 81–96. IEEE Computer Society, 2013.
32. Frank McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *ACM Conference on Management of Data – SIGMOD'09*, pages 19–30. ACM Press, 2009.
33. Daniel Le Métayer. Privacy by design: A formal framework for the analysis of architectural choices. In *ACM Conference on Data and Application Security and Privacy – CODASPY'13*, pages 95–104. ACM Press, 2013.
34. Deirdre K. Mulligan and Jennifer King. Bridging the gap between privacy and design. *University of Pennsylvania Journal of Constitutional Law*, 14:989–1034, 2012.
35. National Institute of Standards and Technology (NIST). MINEXII – an assessment of Match–On–Card technology, 2011. `http://www.nist.gov/itl/iad/ig/minexii.cfm`.
36. Official Journal of the European Union. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation), 2016.
37. International Standard Organization. International standard iso/iec 24787:2010, information technology – identification cards – on-card biometric comparison, 2010.
38. Elena Pagnin, Christos Dimitrakakis, Aysajan Abidin, and Aikaterini Mitrokotsa. On the leakage of information in biometric authentication. In *INDOCRYPT'14*, volume 8885 of *LNCS*, pages 265–280. Springer, 2014.
39. Riccardo Pucella. Deductive algorithmic knowledge. *J. Log. Comput.*, 16(2):287–309, 2006.
40. Koen Simoens, Julien Bringer, Hervé Chabanne, and Stefaan Seys. A framework for analyzing template security and privacy in biometric authentication systems. *IEEE Transactions on Information Forensics and Security*, 7(2):833–841, 2012.
41. Sarah Spiekermann and Lorrie Faith Cranor. Engineering privacy. *IEEE Trans. Software Eng.*, 35(1):67–82, 2009.

42. Vinh-Thong Ta and Thibaud Antignac. Privacy by design: On the conformance between protocols and architectures. In *Foundations and Practice of Security – FPS'14*, volume 8930 of *LNCS*, pages 65–81. Springer, 2015.
43. Qiang Tang, Julien Bringer, Hervé Chabanne, and David Pointcheval. A formal study of the privacy concerns in biometric-based remote authentication schemes. In *Information Security Practice and Experience – ISPEC'08*, volume 4991 of *LNCS*, pages 56–70. Springer, 2008.
44. Juan Ramón Troncoso-Pastoriza and Fernando Pérez-González. Fully homomorphic faces. In *International Conference on Image Processing – ICIP'12*, pages 2657–2660. IEEE Computer Society, 2012.
45. Umut Uludag, Sharath Pankanti, and Anil K. Jain. Fuzzy vault for fingerprints. In *Conference on Audio– and Video–Based Biometric Person Authentication – AVBPA'05*, volume 3546 of *LNCS*, pages 310–319. Springer, 2005.